UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Modelado de pandemias vía Aprendizaje Automático

Máster Universitario en Ingeniería de Telecomunicación

Autor: GÓMEZ HERNÁNDEZ, Belén
Tutor: DOMÍNGUEZ CARRETA, David Renato

Junio 2021

Modelado de pandemias vía Aprendizaje Automático

AUTOR: Belén Gómez Hernández
TUTOR: David Renato Domínguez Carreta

Máster de Ingeniería de Telecomunicación
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2021

Agradecimientos

En primer lugar quiero agradecer a mi familia el apoyo aportado durante mi formación académica, en particular a mis padres Bernabé y Belén, y a mi hermana Sheila. También a todas las personas que me han aconsejado y ayudado en esta etapa de mi vida.

Dar las gracias a mi tutor David, por darme la oportunidad de hacer este TFM.

Y por último, quiero hacer especial mención a los que me vieron comenzar este Máster y lamentablemente no pudieron ser cómplices en vida del final.

RESUMEN

El auge en las tecnologías ha dado paso a la creación de modelos de Inteligencia Artificial con algoritmos capaces de procesar información que de otra manera sería imposible de analizar.

Dentro de estas tecnologías, podemos encontrar algoritmos que simulan el comportamiento del cerebro humano a una escala inferior, ya que el comportamiento real de procesamiento humano aún es inviable con las tecnologías actuales.

El objetivo de este Trabajo de Fin de Máster es aplicar un modelo de aprendizaje automático a un caso de la vida real, para poder dar una solución a un problema social.
El modelo principal que se va a implementar es una red neuronal que nos ayudará a predecir datos para posteriormente realizar un análisis en cuanto a los resultados obtenidos, y también de la optimización del algoritmo que vamos a implementar.

En este proyecto trataremos el problema actual de la pandemia de COVID para intentar crear una conclusión coherente de los datos que analizaremos.
Realizaremos una búsqueda de datos oficiales sobre este tema e intentaremos encontrar correlaciones entre los mismos.

Buscaremos analizar países en base al índice de desarrollo humano para después aplicarles el modelo y comparar los resultados entre ellos.

En definitiva, la intención de este trabajo es implementar un modelo de Inteligencia Artificial a un problema de enfermedades globales para poder generar una hipótesis que nos ayude a tener una mayor comprensión de la evolución de la pandemia.

PALABRAS CLAVE

Machine Learning, *Deep Learning*, Redes Neuronales, Red LSTM, *Keras*, COVID, Países

ABSTRACT

The boom in technologies has given way to the creation of Artificial Intelligence models with algorithms that are able to process information that would otherwise be impossible to analyze.

Within these technologies, we can find algorithms that simulate the behavior of the human brain at a lower scale, since the real human processing behavior is still not viable with current technologies.

The aim of this Master's Thesis is to apply a Machine Learning model to a real life case, in order to provide a solution to a social problem.
The principal model to be implemented is a neural network that will help us to predict data to later perform an analysis of the results obtained, and also the optimization of the algorithm that we are going to implement.

In this project we will deal with the current problem of the COVID pandemic to try to create a coherent conclusion from the data we will analyze.
We will perform a search of official data on this topic and try to find correlations between them.

We will try to analyze countries based on the human development index and then apply the model to them and compare the results between them.

In conclusion, the intention of this work is to implement an Artificial Intelligence model to a global disease problem in order to generate a hypothesis that will help us to have a better understanding of the evolution of the pandemic.

KEYWORDS

Machine Learning, Deep Learning, Neural Networks, LSTM Network, Keras, COVID, Countries

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 *Motivación*

Debido a la crisis sanitaria que se está produciendo y a la gran cantidad de información que se está recopilando a causa de la misma, existe una motivación para realizar un trabajo de fin de máster que nos ayude a aplicar la tecnología con métodos aprendidos en la formación del máster y así poder obtener conclusiones de estos datos.

En concreto, el trabajo de fin de máster llevado a cabo se centra en el modelado de enfermedades, en particular la pandemia de Covid-19.

Para ello se recopilarán datos socioeconómicos, densidad de población, tasa de alfabetización, valor del producto interior bruto y esperanza media de vida [2][5]. Así como los datos propios de Covid-19 por países, distinguiendo incluso por rangos de edad [3][4].

El objetivo de este modelaje de pandemias es buscar un modelo de predicción y una leve correlación entre la enfermedad y, los datos socioeconómicos y de vacunación. A través de herramientas de aprendizaje automático, entre ellas algoritmos genéticos y redes neuronales recurrentes como las LSTM, se medirá su capacidad de predicción de patrones de contagio. Se buscará la optimización del modelo de red neuronal.

Con este trabajo fin de máster se pretende potenciar las siguientes competencias del máster:

- Aplicación de modelos de aprendizaje automático: búsqueda de información, recopilación de datos, preparación de datos para aplicar el modelo (conversión de datos y normalización/estandarización de los mismos) [1][6][7].

- Trabajar la capacidad para seleccionar la metodología adecuada para obtener resultados a partir de información incompleta o limitada, sin realizar modificaciones que afecten significativamente al contexto de los resultados expuestos.

- Búsqueda de conclusiones y análisis de las mismas.

- Trabajar la capacidad de comunicar las conclusiones y conocimientos de una manera clara, concisa y sin ambigüedades a un público especializado o no, justificando lógicamente las decisiones tomadas sin dejar de considerar alternativas o puntos de vista complementarios.

1.2 *Objetivos*

Este trabajo de fin de máster tiene como objetivos buscar correlaciones entre los datos socioeconómicos y epidemiológicos recolectados con el fin de estudiar la tasa de contagios producidos en la pandemia de Covid-19.

Las pautas a seguir para llevar a cabo el trabajo son:

- Documentarse en el área a investigar. En primer lugar, aprender los conceptos básicos de *Machine Learning*.

- Buscar datos de fuentes oficiales y actualizadas, para ello es necesario hacer una búsqueda exhaustiva en bases de datos públicas que contengan información verídica.

- Analizar las metodologías existentes y modelos de *Machine Learning* (redes neuronales recurrentes como las LSTM) que se van a aplicar en base a las características de los datos que disponemos.

- Realizar representaciones gráficas de los resultados para poder tener una visión analítica de la información obtenida después de aplicar el modelo de *Machine Learning* seleccionado.

- Modificar los parámetros de los modelos con el fin de obtener distintos resultados y poder contemplar otros puntos de vista. Mediante la optimización del modelo se pueden tener mejores conclusiones.

- Analizar todos los resultados obtenidos de una manera global, para así poder obtener conclusiones racionales y extraer conocimientos de ellas.

El objetivo de este proyecto reside en la búsqueda de un modelo que aporte conocimientos como poder predecir comportamientos y datos que tienen impacto en la población y sean de utilidad para poder generar planes y estrategias para combatir pandemias como la del COVID.

1.3 *Organización de la memoria*

La memoria consta de los siguientes capítulos:

- **Capítulo 1: Motivación, objetivos y organización de la memoria.**
- **Capítulo 2: Estado del arte.**

  Analizaremos las herramientas con las que vamos a desarrollar el proyecto.
- **Capítulo 3: Diseño.**

  En este capítulo podremos encontrar la hipótesis planteada y el análisis de los datos que se van a utilizar.

- **Capítulo 4: Desarrollo.**

  Aplicaremos los modelos estudiados en el capítulo 2 a los datos preprocesados que se presentan en el capítulo 3.
- **Capítulo 5: Integración, pruebas y resultados.**

  Presentaremos los estudios llevados a cabo para obtener los mejores resultados en el proyecto.
- **Capítulo 6: Conclusiones y trabajo futuro.**

2 Estado del arte

2.1 *Inteligencia artificial*

La inteligencia artificial (IA):

- es el campo de estudio que da a las maquinas (ordenadores) la capacidad de aprender sin ser programados explícitamente para ello, es decir, la habilidad de una máquina para "pensar" como un ser humano.
- permite que los sistemas tecnológicos reciban información del entorno, resuelvan problemas y actúen en base a un aprendizaje y unas capacidades adquiridas.
- contiene el *Machine Learning* (ML) y el *Deep Learning* (DL). [8]

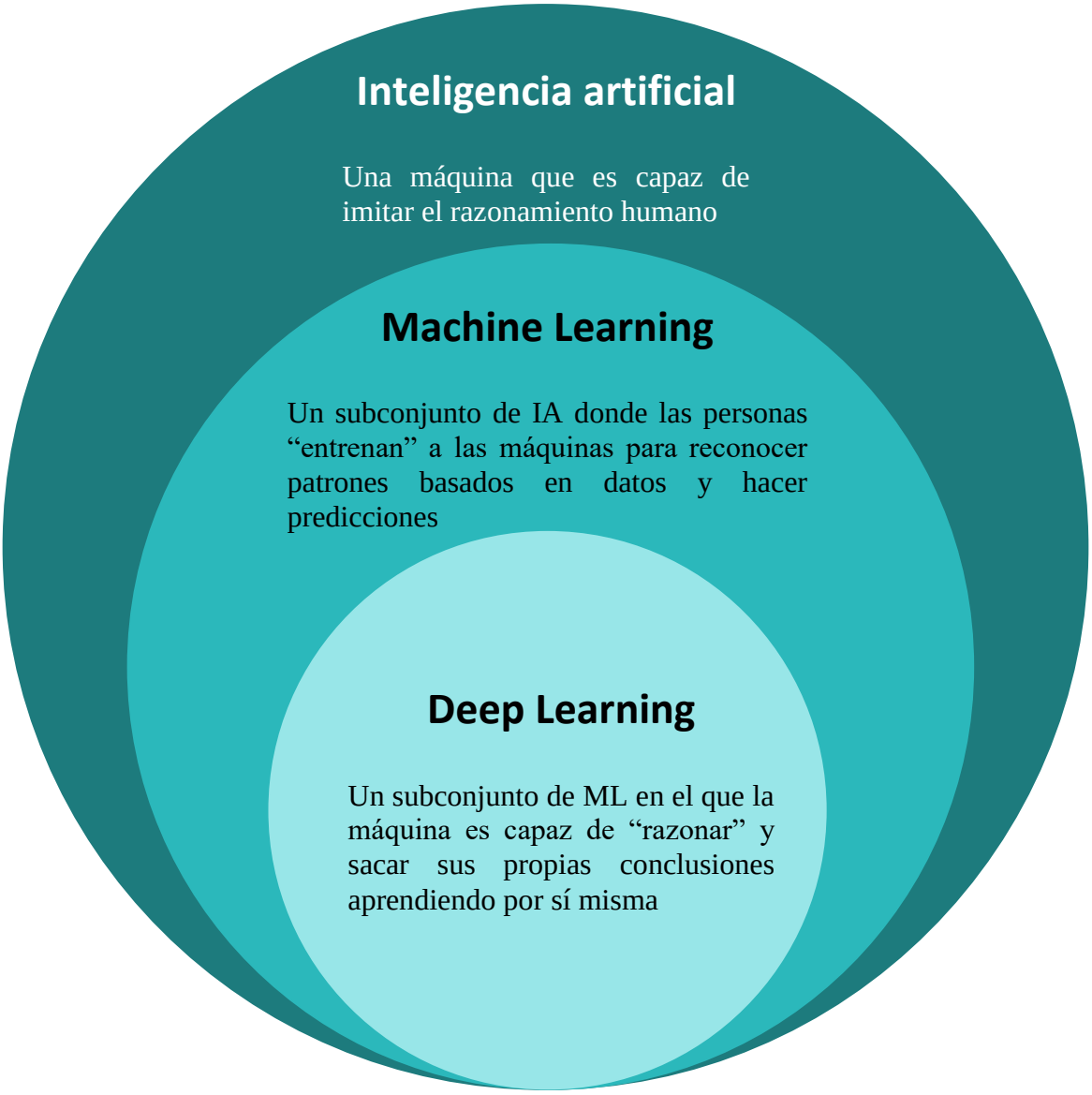


Figura 1 : Estructura Inteligencia Artificial. Fuente: Elaboración propia.

2.1.1 Aprendizaje automático o *Machine Learning*

El *Machine Learning* proporciona a las maquinas la capacidad de resolver problemas sin la necesidad de programar explícitamente la solución. Sin el ML la inteligencia artificial quedaría limitada al hecho de decidir ejecutar un código en base a si algo es verdadero o no, es decir (si X es verdadero, ejecuta Y o de lo contrario, ejecuta Z).

Los tipos de algoritmos de M*achine Learning* más conocidos son:

- **Supervisado:** en el aprendizaje supervisado se introduce al algoritmo un conjunto de datos conocidas las entradas y salidas deseadas, y el algoritmo busca un método (identifica patrones en los datos) para llegar a esas entradas y salidas. El algoritmo realiza predicciones y correcciones en el proceso hasta llegar a un nivel de precisión y rendimiento aceptables.
  Existen dos tipos:
  - **Clasificación:** clasifican los datos dentro de diversas clases.
  - **Regresión:** predicen un valor o un dato.

- **No supervisado:** el algoritmo estudia los datos para identificar patrones, no tiene una respuesta de validación. Sin embargo, el algoritmo determina correlaciones en base al análisis de los datos.
  Existen dos tipos:
  - **Clustering:** clasifican los datos dentro de grupos. Por ejemplo: *k-means.*
  - **Asociación o reducción de la dimensionalidad:** descubre correlaciones (reglas) entre un conjunto de datos. Por ejemplo: PCA (*Principal Component Analysis*).

- **Por refuerzo:** el aprendizaje analiza diferentes opciones, las monitoriza y las evalúa para determinar cuál es el resultado más óptimo. Utiliza el método popularmente conocido como prueba y error. Por ejemplo: *Q-learning*.

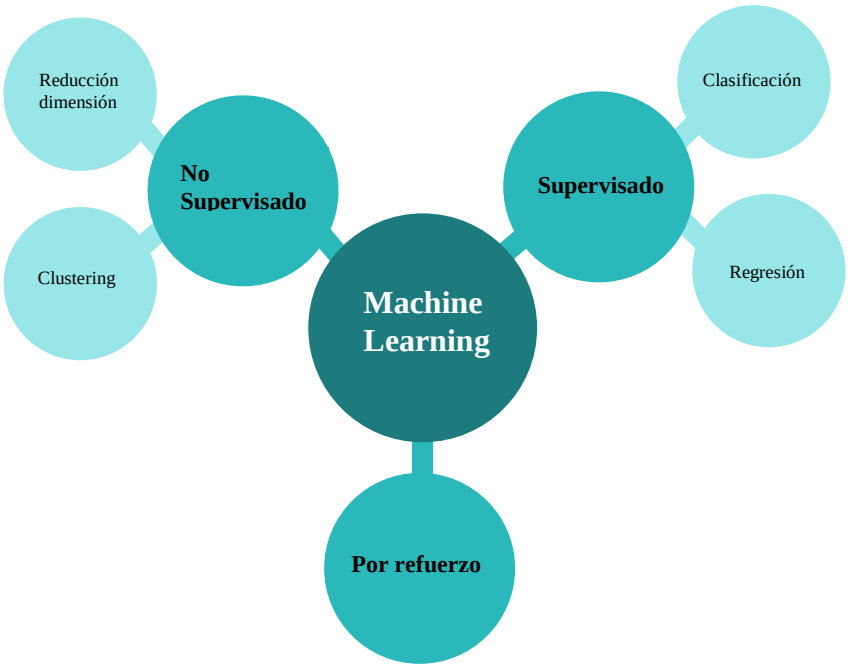


Figura 2 : Estructura *Machine Learning*. Fuente: Elaboración propia

2.1.2 Aprendizaje profundo o *Deep Learning*

El D*eep Learning* es el subconjunto del *Machine Learning* que contiene los algoritmos que pretenden imitar el comportamiento del cerebro humano mediante redes neuronales artificiales. [9]

Los tipos de algoritmos de D*eep Learning* más conocidos son:

- **Redes de perceptrones multicapa o MLP (*Multi Layer Perceptron*):** es una evolución del perceptrón simple que incorpora capas ocultas consiguiendo representar funciones no lineales. El perceptrón multicapa está formado por una capa de entrada, una de salida y n capas ocultas.

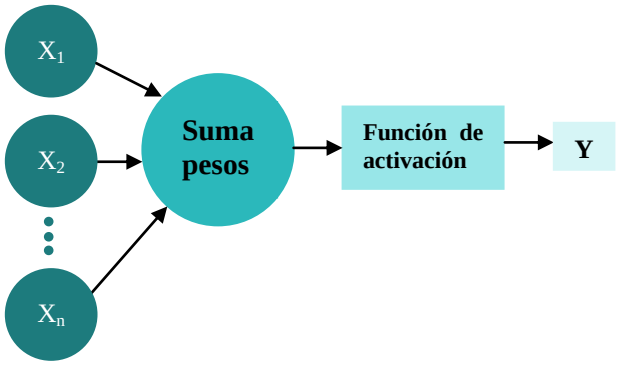


Figura 3 : Diagrama Ejemplo Perceptrón Simple. Fuente: Elaboración propia.

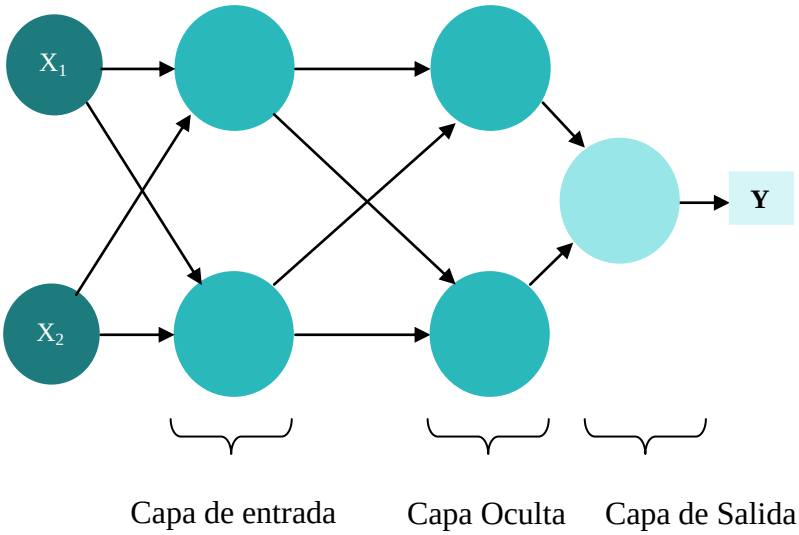


Figura 4 : Diagrama Ejemplo Perceptrón Multicapa. Fuente: Elaboración propia.

En las redes MLP podemos distinguir dos fases:

- **Propagación** (*forward propagation*): en la que se calcula la salida desde los valores de entrada hacia delante. En este proceso cada neurona realiza una suma ponderada de todas las entradas en base a unos pesos, pasando el resultado por una función de activación y generando un resultado que pasa a la siguiente capa.

- **Aprendizaje:** en la que los errores a la salida del perceptrón se propagan hacia atrás (*backpropagation*) con el objetivo de modificar los parámetros de las neuronas para conseguir un resultado más veraz.[10]

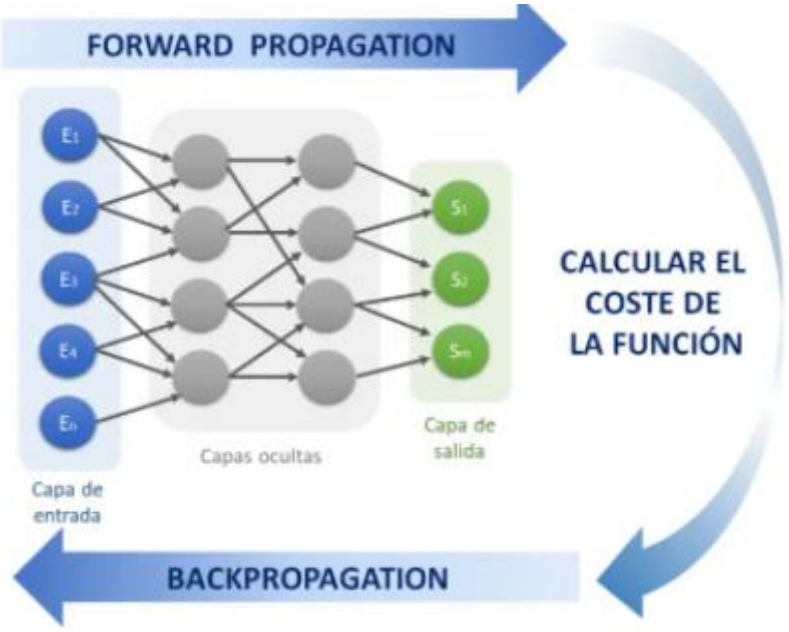


Figura 5 : Propagación y aprendizaje. Fuente: Referencia 10.

- **Redes neuronales convolucionales o CNN (*Convolutional Neural Network*):** a diferencia del perceptrón multicapa, las redes neuronales convolucionales no une todas las neuronas de una capa con todas las de la siguiente, sino que solo lo hace con un subgrupo logrando reducir el número de neuronas necesarias y la complejidad computacional que conlleva la ejecución de la red. [11]

La arquitectura básica de una red convolucional utiliza los siguientes elementos:

- **Entrada:** este tipo de redes es el más utilizado en procesamiento de imágenes por lo que normalmente la entrada suelen ser los pixeles de la imagen.

- **Capa de convolución:** calcula el producto escalar entre los pesos de las neuronas de salida que están conectadas a las de entrada.

- **Capa ReLU (*Rectified Linear Unit*):** aplica la función de activación lineal rectificadora que desactiva las neuronas con salida negativa.

- **Capa *Pooling*:** realiza una reducción de las dimensiones, pero mantiene la profundidad.

- **Capa *Softmax*:** asigna probabilidades de los datos de pertenecer a distintas clases posibles.

- **Capa *Fully-connected*:** en esta capa todas las neuronas están conectadas a todas las neuronas de la siguiente capa.

Un ejemplo de red neuronal convolucional sería:

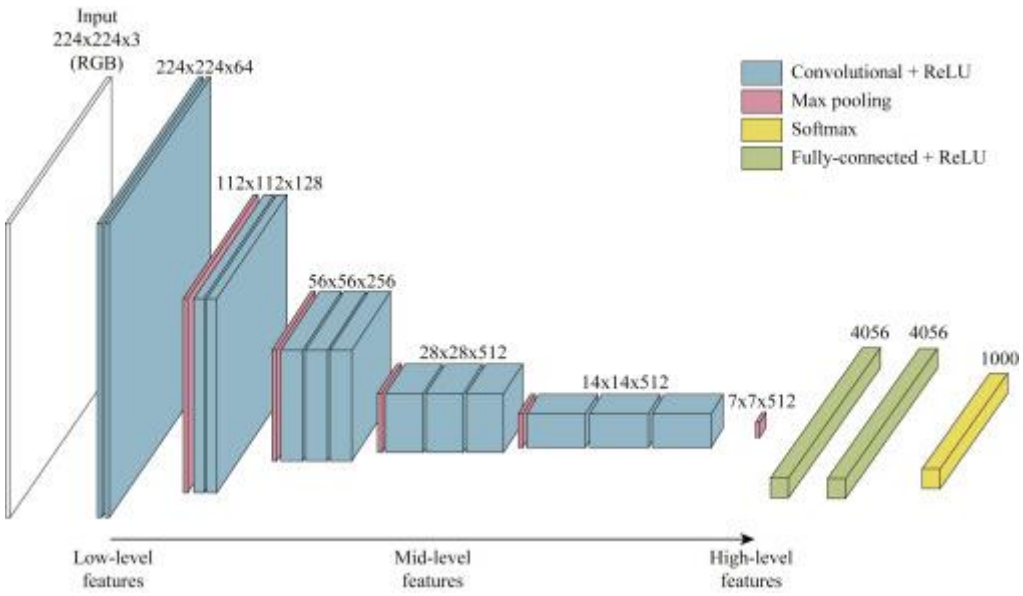


Figura 6 : Ejemplo Red Neuronal Convolucional. Fuente: Referencia 12.

- **Redes neuronales recurrentes o RNN (*Recurrent Neural Network*):** estas redes integran bucles de realimentación, permitiendo a través de ellos que la información persista durante algunos pasos o épocas de entrenamiento (*epochs*), a través de conexiones desde las salidas de las capas, que incrustan sus resultados en los datos de entrada. [13]

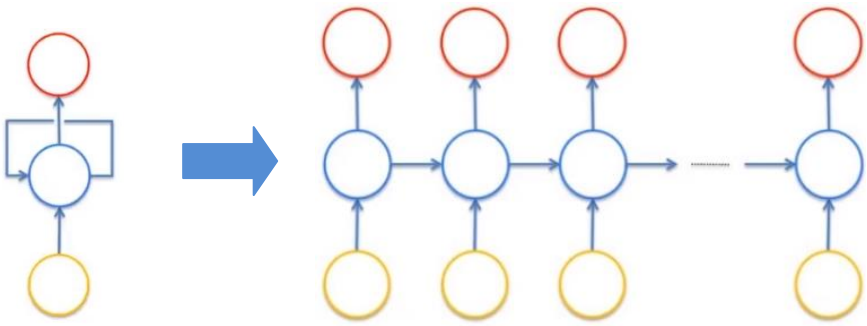


Figura 7: Ejemplo Red Neuronal Recurrente. Fuente: Elaboración propia.

2.2 *Herramientas*

Para realizar el estudio con Inteligencia Artificial, vamos a necesitar ciertas herramientas que nos faciliten las implementaciones de los modelos y que nos permitan analizar y visualizar los datos.

2.2.1 Programas y lenguaje de programación

La Inteligencia Artificial necesita lenguajes de programación que puedan manejar las estructuras y funciones requeridas. Por ello, *Python* es el lenguaje más extendido entre los programadores de *Machine Learning.* Y es que *Python* ofrece multitud de librerías como parte de su esencia como código abierto, además es un lenguaje flexible, con esto queremos decir que es fácilmente combinable con otros tipos de lenguajes de programación.
Con *Python* es posible encontrar una solución de visualización que se adapte perfectamente a cualquier proyecto, y la potencia de sus herramientas de visualización ofrece un gran detalle en el análisis de los datos y la detección de errores. [14]

A continuación, explicamos los dos programas utilizados para programar en *Python* el proyecto:

- *Google Colaboratory*
- *Spyder*

2.2.1.1 Google Colaboratory

Google Colaboratory o como mayormente es conocido "*Colab*", es una herramienta que te permite ejecutar y programar en Python en el navegador con las ventajas de que no requiere de ninguna configuración, es de acceso gratuito a GPUs y permite compartir el contenido de una forma sencilla. [15]

Este programa es ideal para realizar trabajos de IA, ya que proporciona un entorno que permite escribir y ejecutar código (los cuadernos de *Colab*, son cuadernos de *Jupyter* almacenados en la nube).

Los cuadernos de *Colab* permiten combinar código ejecutable y texto, además también podemos incluir imágenes, HTML, LaTeX y muchas más cosas. Estos cuadernos se almacenen en la cuenta de *Google Drive* y se pueden compartir de manera que sea editable simultáneamente por todos los usuarios que tengan el archivo compartido. [16]

Figura 8 : Google Colaboratory. Fuente: Google Imágenes.

Con *Colab* tenemos acceso a múltiples bibliotecas de Python para analizar y visualizar datos como *Pandas*, *Numpy*, *Matplot*…

2.2.1.1 Spyder

Spyder (*Scientific Python Development Environment*) es un entorno de desarrollo interactivo para Python. Tiene funciones avanzadas de edición y depuración. Es un entorno de desarrollo integrado y multiplataforma de código abierto (IDE: *Integrated Development Environment*), accesible a través de Anaconda.

Figura 9 : Spyder y Anaconda. Fuente: Google Imágenes

Para la realización de este TFM se han utilizado tanto *Google Colaboratory* como *Spyder*, ya que el análisis de los tipos de datos tiene mayor visibilidad en *Spyder*, mientras que para interactuar con el tutor, se requería de una herramienta que permitiera dinamismo como es *Google Colaboratory*.

2.2.2 Librerías necesarias

En este proyecto hemos trabajado con los siguientes paquetes y librerías:

***Tensorflow*:** es una librería desarrollada por Google que es de código abierto desde noviembre de 2005.
Es capaz de implementar redes neuronales considerablemente grandes de manera eficiente. Esta librería fue diseñada para computación numérica de gran velocidad, y es que se trata de una librería que computa gradientes automáticamente, esto quiere decir que está a un nivel más bajo y profundo que otras librerías como *Keras*.

Keras: es una librería de alto nivel desarrollada en *Python* por *François Chollet*.
Se trata de una librería de código abierto concebida para ejecutarse sobre *Tensorflow*, *Microsoft Cognitive Toolkit* o *Theano*. Ha sido diseñada específicamente para experimentos con redes neuronales y permite crear prototipos de una manera sencilla.

***Scikit-Learn*:** es una biblioteca de código abierto que provee herramientas científicas para el análisis de datos y *data mining* (minería de datos). Emplea algoritmos de clasificación, regresión y agrupamiento; y opera de manera simultánea con librerías como *NumPy*. [17]

***Numpy*:** es una librería de *Python* orientada al cálculo vectorial y matricial. Proporciona una estructura de datos de matriz que garantizan cálculos eficientes con matrices. [18]

***Pandas*:** es una librería de *Python* concebida como una extensión de *NumPy* que permite el procesamiento de tablas, series temporales y otras estructuras complejas de datos.

***Matplotlib*:** es una librería específica para la generación de gráficos en *Python* a partir de datos contenidos en listas o *arrays*. Puede generar histogramas, gráficos de barras, de errores, de dispersión, etc.

Además de las librerías mencionadas, también importamos el módulo *random* que genera números pseudoaleatorios para varias distribuciones.

Las librerías que se utilizarán en el proyecto son:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
import keras
import sklearn
import random
```

3 Diseño

3.1 Diagrama etapas de Machine learning

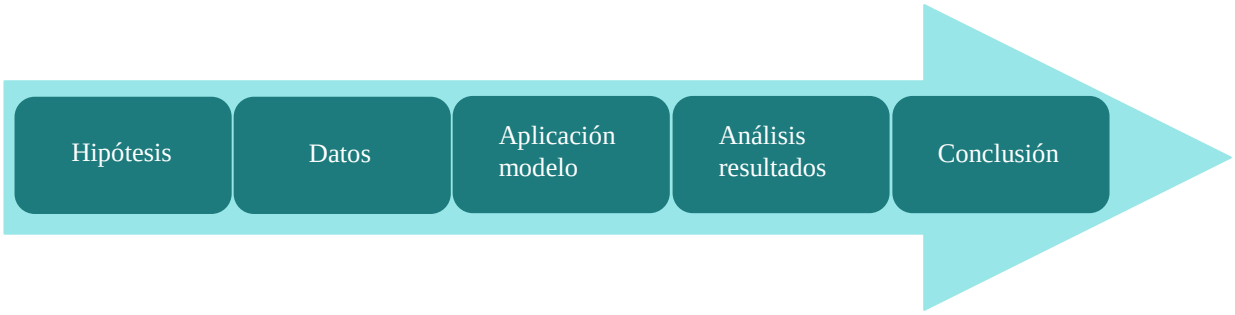


Figura 10 : Etapas Machine Learning. Fuente: Elaboración propia.

Todo proyecto de *Machine Learning* realiza un proceso para llegar a unas conclusiones. Las siguientes etapas son las que seguiremos para realizar el proyecto:

- **Hipótesis del problema:** formularemos una hipótesis principal la cuál será nuestra idea base para definir el objetivo y seguir con las siguientes etapas.
- **Lectura y análisis de datos:** recolectaremos y prepararemos los datos necesarios para pasar a nuestro modelo de *Machine Learning*.
- **Aplicación del modelo:** elegiremos el algoritmo que vamos a aplicar, en este caso realizaremos una Red Neuronal. Entrenaremos el modelo y lo validaremos con datos de test.
- **Análisis de los resultados:** Una vez tengamos los resultados del modelo, analizaremos si nuestro modelo cumple el objetivo.
- **Conclusión:** con los resultados y análisis de los resultados formularemos una conclusión del proceso.

3.2 Hipótesis del problema

Nuestra hipótesis principal es que a medida que un país está más desarrollado, es decir, presenta un mayor índice de desarrollo, presenta una mejor evolución de la pandemia porque está más preparado para tomar medidas y desarrollar estrategias contra la expansión del virus.

Además, aquellos países que contemplen características superiores o inferiores en cuanto a desarrollo (PIB, esperanza de vida y tasa de alfabetización) deben presentar comportamientos (por ejemplo de estrategia de vacunación) similares frente a la pandemia.

Por otro lado, se trabajará desde la hipótesis de que a partir de los datos históricos que tengamos de la evolución de la pandemia, podremos hacer una predicción de los futuros datos. Es decir, con los casos de COVID de las primeras semanas desde el inicio de la expansión de la enfermedad, podremos predecir los casos que habrá en un país distinguiendo incluso por rango de edad en las próximas semanas.

3.3 Datos

Para poder hacer un estudio de la pandemia de COVID, hemos decidido hacer una valoración de los países más “desiguales”.

El criterio seleccionado para elegir los países a estudiar en detalle, ha sido mediante el Índice de Desarrollo Humano (IDH).

Posteriormente, hemos analizado una base de datos relacionada con el COVID distinguiendo los casos nuevos por rangos de edad. Además, para poder realizar una valoración del proyecto, hemos analizado datos del proceso de vacunación.

3.4 Selección de Bases de Datos

Uno de los objetivos de este TFM era aportar información real, obtenida de fuentes verificadas y oficiales. Por ello, la búsqueda de información ha sido una de las tareas más complicadas a llevar a cabo, y es que hoy en día podemos encontrar numerosas fuentes de información pero es complicado dar con la adecuada.

Buscamos fuentes de información oficiales que nos puedan aportar los datos en formato CSV (C*omma-Separated Values*) para mayor facilidad a la hora de trabajar con ellos en *Python*.

En nuestro caso, hemos decidido optar por las siguientes fuentes oficiales:

- *World Bank* (WB) o Banco Mundial.
- *European Centre for Disease Prevention and Control* (*ECDC*) o Centro Europeo para la Prevención y el Control de las Enfermedades.

Figura 11 : Fuentes Bases de datos. Fuente: Google Imágenes.

3.4.1 *World Bank* (WB) o Banco Mundial. Datos IDH

El IDH es un indicador de progreso que fue elaborado por el Programa de las Naciones Unidas para el Desarrollo (PNUD); es utilizado para clasificar los países en niveles de desarrollo humano. El índice se calcula a partir de los factores determinantes de PIB (Producto Interior Bruto) por habitante, esperanza de vida y tasa de alfabetización. [19]

Para un determinado país la forma de calcular cada índice de dimensión utiliza la siguiente fórmula:

Índice de dimensión = (valor actual – valor mínimo)/(valor máximo – valor mínimo)

Una vez calculados los índices de cada dimensión, se realiza el cálculo del índice final:

$$IDH = (I_{PIB} * I_{esperanza\ de\ vida} * I_{tasa\ alfabetización})^{1/3}$$

Como se puede ver en la fórmula del IDH, los tres factores tienen la misma importancia en ella, por lo que nuestro primer objetivo es el estudio de la relevancia de los factores.

La base de datos del Banco Mundial tiene una recopilación de estadísticas relevantes, de alta calidad e internacionalmente comparables sobre el desarrollo mundial y la lucha contra la pobreza. Contiene 1400 indicadores de series de tiempo para 217 países, con datos que se remontan a más de 50 años.

En nuestro caso realizamos un filtrado de 3 indicadores de serie para los 27 países de la Unión Europea en el año 2020.

Como hemos visto anteriormente, los factores determinantes del índice de desarrollo humano son: PIB per cápita, esperanza de vida y tasa de alfabetización. Por lo que nuestros indicadores de serie serán estos y realizaremos el estudio del año 2020 (año de inicio de la pandemia del COVID). [2]

En resumen, de la página oficial del Banco Mundial, obtenemos un archivo CSV con el siguiente formato:

```
,país_Name,país_Code,serie_Name,serie Code,YR2020
0,Austria,AUT,"PIB per cápita, PPA ($ a precios internacionales constantes de 2011)",NY.GDP.PCAP.PP.KD,55630.6084761777
1,Austria,AUT,"Esperanza de vida al nacer, total (años)",SP.DYN.LE00.IN,81.6926829268293
2,Austria,AUT,"Tasa de alfabetización, total de adultos (% de personas de 15 años o más)",SE.ADT.LITR.ZS,..
3,Bélgica,BEL,"PIB per cápita, PPA ($ a precios internacionales constantes de 2011)",NY.GDP.PCAP.PP.KD,51298.6303235428
4,Bélgica,BEL,"Esperanza de vida al nacer, total (años)",SP.DYN.LE00.IN,81.5951219512195
5,Bélgica,BEL,"Tasa de alfabetización, total de adultos (% de personas de 15 años o más)",SE.ADT.LITR.ZS,..
```

Figura 12: Muestra de base de datos WB. Fuente: Referencia 2.

3.4.2 *European Centre for Disease Prevention and Control* (ECDC) o Centro Europeo para la Prevención y el Control de las Enfermedades. Datos Covid y vacunación

El ECDC es una organización que expone datos oficiales sobre enfermedades globales. Teniendo en cuenta que el principal factor que determina el pronóstico de la enfermedad es la edad del enfermo (dada la información generada y los informes de las tasas de incidencia de la enfermedad, se ha demostrado que la mortalidad por debajo de los 50 años ha sido muy baja, pero la mortalidad se incrementaba exponencialmente a partir de esa edad) vamos a hacer uso de los datos que nos proporciona el ECDC distinguiendo por rangos de edad.

Para poder hacer una predicción de los nuevos contagios, vamos a aplicar una red neuronal a estos datos y así predecir por rangos de edad la evolución de la pandemia.

Desde el ECDC se analizan e interpretan los datos sobre 52 enfermedades transmisibles y sobre brotes de enfermedades y amenazas para la salud pública a través del Sistema Europeo de Vigilancia (TESSy).

En nuestro caso, vamos a hacer uso de los datos disponibles sobre COVID ya que contienen datos en función de la edad, variable que es objeto de estudio en este trabajo de fin de máster. Así como los datos de la vacunación. [3]

En resumen, de la página oficial del ECDC, obtenemos un archivo CSV con el siguiente formato:

```
country,country_code,year_week,age_group,new_cases,population,rate_14_day_per_100k,source
Austria,AT,2020-09,<15yr,0,1278692,,"TESSy COVID-19, national weekly data"
Austria,AT,2020-10,<15yr,0,1278692,,"TESSy COVID-19, national weekly data"
Austria,AT,2020-11,<15yr,10,1278692,0.8,"TESSy COVID-19, national weekly data"
Austria,AT,2020-12,<15yr,58,1278692,5.3,"TESSy COVID-19, national weekly data"
Austria,AT,2020-13,<15yr,156,1278692,16.7,"TESSy COVID-19, national weekly data"
Austria,AT,2020-14,<15yr,104,1278692,20.3,"TESSy COVID-19, national weekly data"
```

Figura 13: Muestra de base de datos ECDC COVID. Fuente: Referencia 3.

Esta base de datos contienen información en un intervalo temporal que abarca desde finales de Enero de 2020 (principios de la pandemia) a mediados de Mayo de 2021 (fecha mas actual que podemos recoger).

Los parámetros de la base de datos son los siguientes:

- ***country:*** País.
- ***country_code:*** Codigo ISO (código estandarizado de dos letras) del país.
- ***year_week***: año-semana del año.
- ***age_group:*** grupo de edad.
- ***new_cases:*** número de casos nuevos en COVID.
- ***population:*** población del rango de edad correspondiente.
- ***rate_14_day_per_100k:*** tasa de COVID por cada 100000 personas.
- ***source:*** fuente de datos.

En cuanto a los datos de vacunación, el ECDC nos ofrece la siguiente base de datos en formato CSV:

```
YearWeekISO,FirstDose,FirstDoseRefused,SecondDose,UnknownDose,NumberDosesReceived,Region,Population,ReportingCountry,TargetGroup,Vaccine,Denominator
2020-W53,111,0,0,0,61425,AT,8901064,AT,Age18_24,COM,696064
2020-W53,0,0,0,0,0,AT,8901064,AT,Age18_24,AZ,696064
2020-W53,0,0,0,0,0,AT,8901064,AT,Age18_24,JANSS,696064
2020-W53,0,0,0,0,0,AT,8901064,AT,Age18_24,MOD,696064
2020-W53,0,0,0,0,0,AT,8901064,AT,Age25_49,AZ,3006228
2020-W53,0,0,0,0,0,AT,8901064,AT,Age25_49,JANSS,3006228
2020-W53,1282,0,0,0,61425,AT,8901064,AT,Age25_49,COM,3006228
2020-W53,0,0,0,0,0,AT,8901064,AT,Age25_49,MOD,3006228
2020-W53,980,0,0,0,61425,AT,8901064,AT,Age50_59,COM,1396889
2020-W53,0,0,0,0,0,AT,8901064,AT,Age50_59,JANSS,1396889
2020-W53,0,0,0,0,0,AT,8901064,AT,Age50_59,MOD,1396889
2020-W53,0,0,0,0,0,AT,8901064,AT,Age50_59,AZ,1396889
2020-W53,514,0,0,0,61425,AT,8901064,AT,Age60_69,COM,1016016
2020-W53,0,0,0,0,0,AT,8901064,AT,Age60_69,AZ,1016016
2020-W53,0,0,0,0,0,AT,8901064,AT,Age60_69,MOD,1016016
2020-W53,0,0,0,0,0,AT,8901064,AT,Age60_69,JANSS,1016016
```

Figura 14: Muestra de base de datos ECDC Vacunación. Fuente: Referencia3.

Esta base de datos contienen información en un intervalo temporal que abarca desde finales de Diciembre de 2020 (principio de vacunación a la población) a mediados de Mayo de 2021 (fecha mas actual que podemos recoger).

Los parámetros de la base de datos son los siguientes:

- ***YearWeekISO:*** año-semana del año
- ***FirstDose:*** número de vacunas de primera dosis administradas.
- ***FirstDoseRefused:*** número de personas que rechazan la primera dosis de la vacuna.
- ***SecondDose:*** número de vacunas de segunda dosis administradas.
- ***UnknownDose:*** número de vacunas administradas que no especifica la marca.
- ***NumberDosesReceived:*** número de dosis distribuidas.
- ***Region Population:*** Codigo ISO región del país.
- ***ReportingCountry:*** Codigo ISO país.
- ***TargetGroup:*** grupo de edad.
- ***Vaccine:*** marca de la vacuna administrada.
- ***Denominator:*** población del rango de edad correspondiente.

3.5 Bases de datos incompletas.

El estudio de bases de datos incompletas, afecta al resultado cuando se presenta falta de información. La precisión en los resultados que obtengamos, estará condicionada al porcentaje de valores con pérdidas.

Actualmente, este problema es considerado fundamental ya que su presencia es permanente, por lo que debemos analizar medidas para resolverlo. La elección del método de resolución debe tener en cuenta las pérdidas, el tipo de variables (información) que estamos trabajando, así como el procedimiento que se va a aplicar y los resultados que deseamos obtener.

Las bases de datos que hemos encontrado, no están completas al 100%, y faltan algunos valores, por lo que hemos decido añadirlos de la siguiente manera:

- Calculamos el valor promedio de cada tipo de datos.
- Completamos los campos que faltan con el valor promedio.

También existen otros métodos para sustituir los campos inexistentes como reemplazarlos por los valores de los campos vecinos, interpolando con el resto de datos, asignando un valor fijo o simplemente eliminándolos del *dataframe*. [20]

Por ejemplo, para la base de datos mencionado en el apartado 3.4.2 de esta memoria, obtenemos los siguientes resultados al aplicar este método:

```
Campos totales en Base de datos: 10512

Campos con valores de tipo NaN:
            country                   0
            country_code              0
            year_week                 0
            age_group                 0
            new_cases                 0
            population                0
            rate_14_day_per_100k    156

Porcentaje de valores que se van a reemplazar:
            country                 0.000000
            country_code            0.000000
            year_week               0.000000
            age_group               0.000000
            new_cases               0.000000
            population              0.000000
            rate_14_day_per_100k    1.484018
```

En este caso solo el 1.48% de los valores de la columna "*rate_14_day_per_100k*" van a reemplazarse por el valor promedio, ya que solo 156 de los 10512 campos no tienen la información.

Y, para la base de datos mencionada en el apartado 3.4.1 debemos reemplazar el 8% de los valores de alfabetización por el promedio.

Para llegar a estos resultados primero calculamos el número de campos de nuestro *dataframe* con la función "*shape*". Después contamos el número de campos que tienen información con la función "*count*", y calculamos los campos que no tienen información restando los campos con información al número total de campos.

Por último hayamos el porcentaje de datos que se van a reemplazar por si este valor fuera lo suficientemente relevante para tenerlo en cuenta en nuestros resultados y poder llegar a unas mejores conclusiones. En nuestro caso, al tener un porcentaje ínfimo, damos por buenos los resultados que obtengamos más adelante.

```
# Calcular el porcentaje de datos que se van a estimar
valores_tot = data_frame.shape[0]
print('Campos totales en Base de datos:', valores_tot)

valores_No_NaN = data_frame.count()
valores_NaN = valores_tot - valores_No_NaN

print('Campos con valores de tipo NaN::\n', valores_NaN)

Porcentaje = (valores_NaN/valores_tot)*100
print('Porcentaje de valores que se van a reemplazar:', Porcentaje)
```

4 Desarrollo

4.1 Elección de países bajo estudio.

Partimos de la base de datos del WB, y adaptamos la base de datos a un formato con el que podamos trabajar con mayor facilidad, mostramos los atributos en columnas y eliminamos la columna que no tiene ninguna relevancia en este estudio: "`serie Code`". Además, cambiamos las variables de la columna "`serie_Name`" para tener una abreviatura más fácil de manejar:

- `Esperanza de vida al nacer, total (años)` -> `esperanza`
- `Tasa de alfabetización, total de adultos (% de personas de 15 años o más)` -> `alfabetización`
- `PIB per cápita, PPA ($ a precios internacionales constantes de 2011` -> `pib`

Después de procesar los datos, tenemos un conjunto de valores ordenado de la siguiente forma:

país_Name	país_Code	pib	esperanza	alfabetizacion
Austria	AUT	55630.6	81.6927	84.424
Bélgica	BEL	51298.6	81.5951	84.424
Chipre	CYP	38822.5	80.828	84.424
Croacia	HRV	27668.8	78.0707	84.424
Dinamarca	DNK	56444.2	80.9537	84.424
Eslovenia	SVN	38138.9	81.378	84.424
España	ESP	40359.7	83.4317	98.4365

Figura 15: Base de datos procesada. Fuente: Elaboración propia.

Representamos los datos para poder elegir aquellos que creamos más interesantes poner bajo estudio:

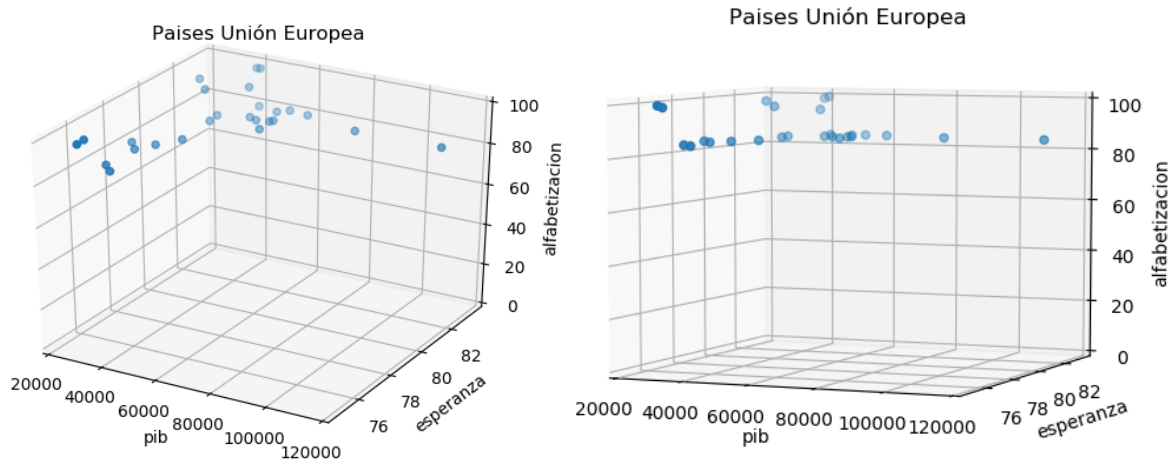


Figura 16: Representación 3 factores IDH. Fuente: Elaboración propia.

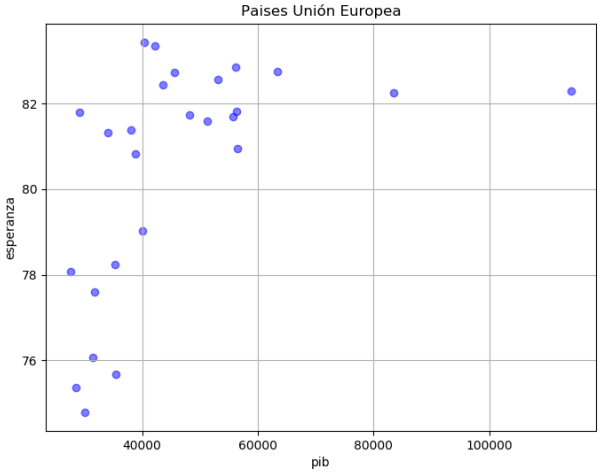


Figura 17: Representación 2 factores IDH. Fuente: Elaboración propia.

Al analizar la representación de los factores, podemos ver que la mayoría de los países de la unión Europea presentan una alta alfabetización, por lo que hacer un estudio comparativo basándonos en este factor no nos proporcionaría ninguna información relevante en el estudio.
Dicho esto, analizaremos los países que consideramos más significativos en base a la esperanza de vida (factor muy importante en ese proyecto ya que está muy relacionado con la salud de las personas), y el producto interior bruto del país (este factor nos puede dar una ligera idea de la "riqueza" que posee un país, lo que suele estar bastante relacionado con la calidad de vida).

Recordamos que durante la pandemia hemos visto como aquellos países con más "riqueza" han podido obtener mayor cantidad de materiales para poder luchar contra la enfermedad, así que los países con una economía superior han tenido mejores condiciones para disminuir los casos activos de COVID.

En conclusión, los países bajo estudio serán aquellos con mayor PIB (Luxemburgo), menor PIB (Croacia), mayor esperanza de vida (España) y menor esperanza de vida (Letonia).

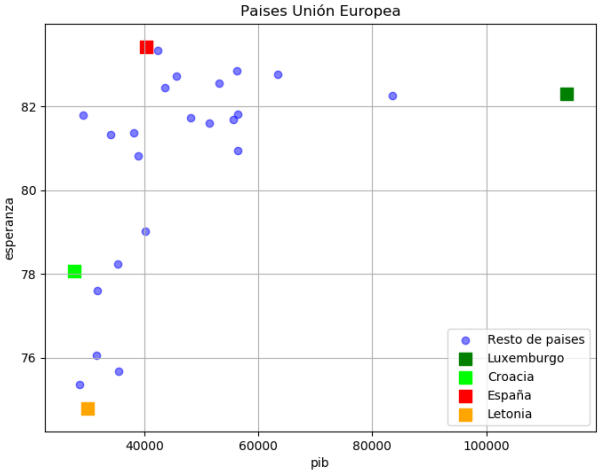


Figura 18 : Paises bajo estudio. Fuente: Elaboración propia.

4.2 Aplicación Redes Neuronales en datos COVID

A continuación, vamos a explicar la aplicación de una Red Neuronal Recurrente o *Recurrent Neural Network* (RNN) para predecir un valor futuro. En concreto, vamos a aplicar una Red LSTM.

Primero vamos a explicar en detalle qué es una Red Neuronal Recurrente, y después una red LSTM y por qué hemos decidido implementarla en nuestra base de datos.

4.2.1 Red Neuronal Recurrente o *Recurrent Neural Network* (RNN)

La mayor debilidad de las redes neuronales tradicionales es que no tienen la capacidad de persistencia en la memoria, por lo que la información aprendida con anterioridad, no presenta efectos en los resultados de la red. Esto es una debilidad ya que las redes neuronales buscan simular el comportamiento de la memoria humana, y esta sí que es capaz de aprender en base a la información ya aprendida.

Para resolver este problema de pérdida de información existen las redes neuronales recurrentes, que gracias a su topología de bucle son capaces de tener persistencia de información

Una Red Neuronal Recurrente básica tiene dos entradas: el dato actual y el estado oculto anterior. Y nos proporciona dos salidas: la predicción y el valor actualizado del estado oculto.

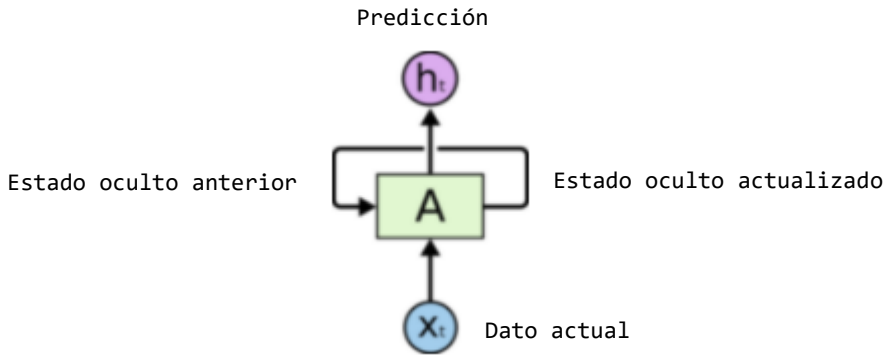


Figura 19 : RNN. Fuente: Referencia 22.

El concepto que maneja la red recurrente es analizar una secuencia de datos de entrada (x_t) y producir una prediccion a la salida (h_t). Para que esto se cumpla, lo primero que toma la red es el estado oculto anterior y la entrada (x_t), y genera un nuevo estado oculto. Después, genera la predicción tomando este nuevo estado oculto.

Este procedimiento puede parecer algo confuso, pero debemos contemplar las redes recurrentes como concatenaciones de una misma red neuronal. El bucle permite que la información pase de un ciclo o bloque de la red al siguiente.

Si “desenrrollamos” la red neuronal recurrente, obtendriamos la siguiente concatenacion de la misma red:

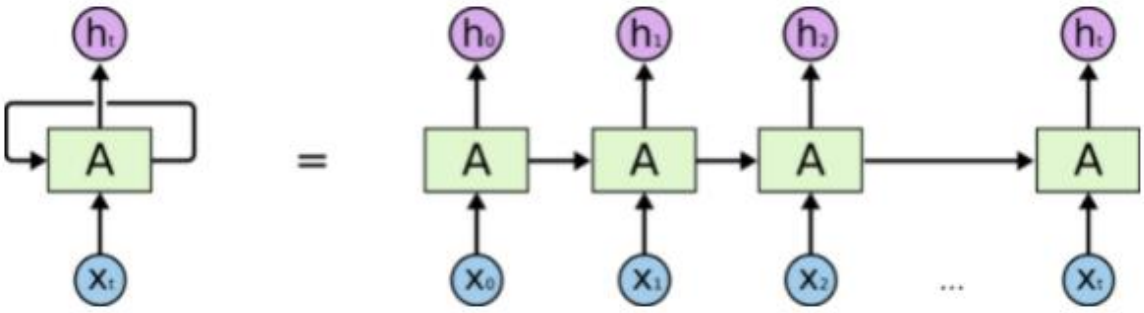


Figura 20 : RNN desenrollada. Fuente: Referencia 22.

El uso de este tipo de redes, esta asociado a la idea de poder conectar la información actual con la anterior; y esta forma de cadena que tiene la topologia hace que las redes estén relacionadas con secuencias y listas.

Este tipo de redes es muy utilizado en reconocimiento y modelacion de textos, traducciones, analisis de imágenes…

En teoria las RNN son capaces de manejar datos a largo plazo, sin embargo a veces solo necesitamos aprender 'dependencias' temporales a largo plazo. Para ello, las redes LSTM nos proporcionan la capacidad de mantener información en la memoria durante largos periodos de tiempo.

4.2.2 Red Neuronal LSTM (*Long short-term memory*)

Una red neuronal LSTM (*Long short-term memory*) es un tipo de Red Neuronal Recurrente que permite analizar secuencias de comportamiento histórico y, además tiene memoria a largo plazo.
Las redes LSTM están diseñadas explícitamente para evitar el problema de la dependencia a largo plazo.

Todas las redes neuronales recurrentes tienen la forma de una cadena de módulos que repiten la red neuronal. En las RNN tradicionales el módulo tiene una estructura muy simple como una sola capa de *tanh*. [23]

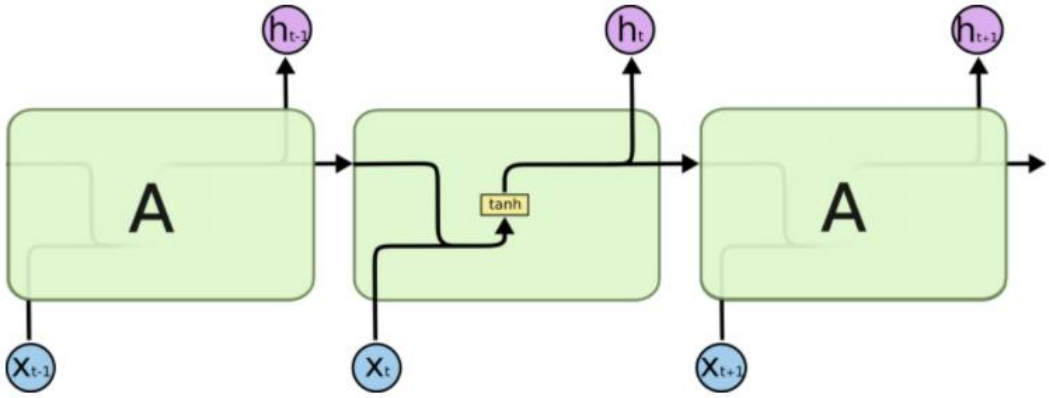


Figura 21 : Módulo de RNN con una sola capa. Fuente: Referencia 22.

Las LSTM también tienen la estructura en cadena, sin embargo el módulo tiene una estructura diferente. En lugar de tener una sola capa, hay cuatro.

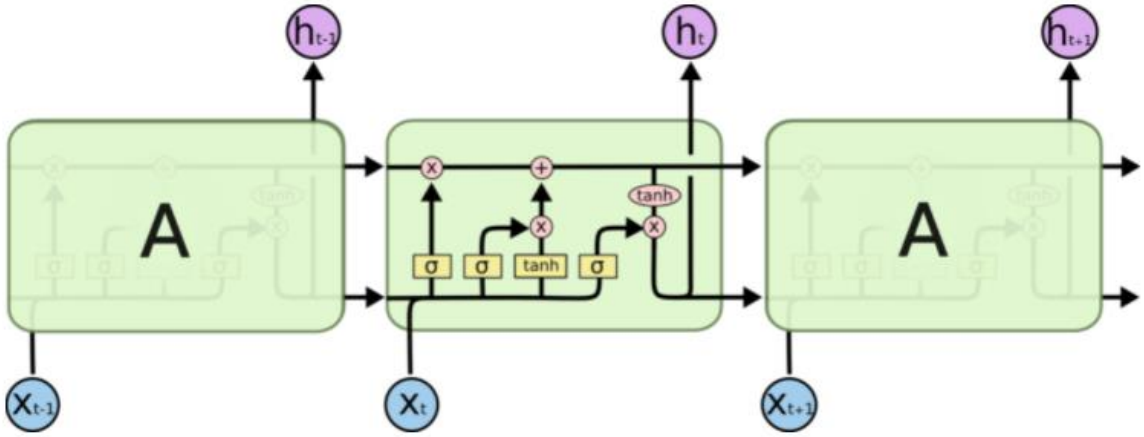


Figura 22 : Módulo de LSTM. Fuente: Referencia 22.

En el diagrama anterior, cada línea lleva un vector desde la salida de un nodo hasta las entradas de otros. Los círculos rosas representan operaciones como la suma o multiplicación de vectores, los cuadros amarillos representan las capas de redes neuronales y, las líneas denotan concatenación y copia de vectores.

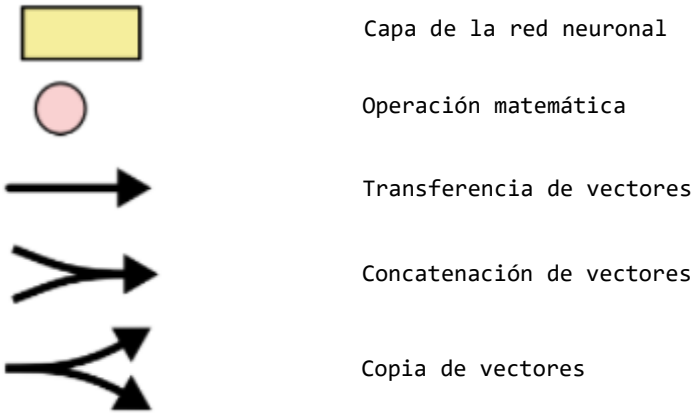


Figura 23 : Notación diagrama de red LSTM. Fuente: Referencia 22.

Para explicar el funcionamiento de la red LSTM nos vamos a centrar en un bloque, donde tenemos **C** que es la memoria de nuestra red (lo que llamaremos "estado"), **h** que sería la información de salida (la predicción) y **X** que sería la entrada de la red.

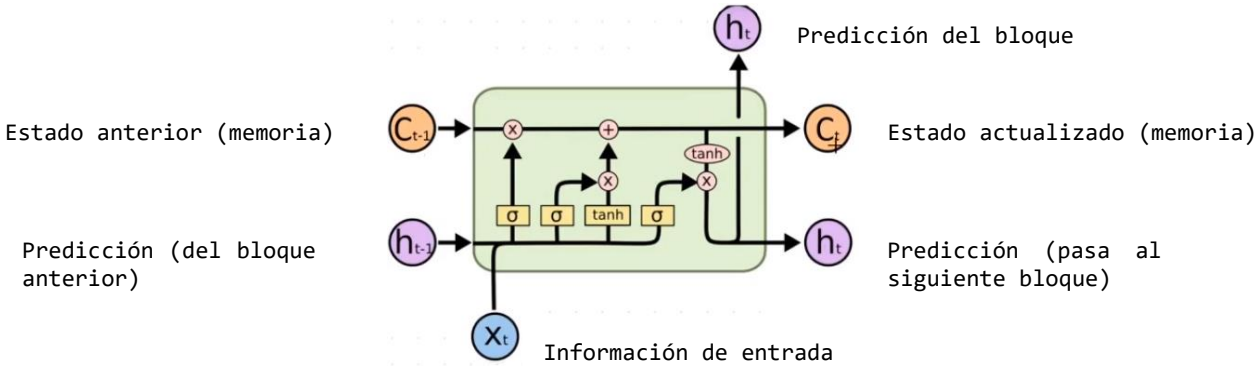


Figura 24: Bloque de Red LSTM. Fuente: Referencia 22.

La clave de las redes LSTM está en el estado del módulo. La línea horizontal superior pasa a lo largo de la cadena el estado del módulo tan solo haciendo algunas iteraciones lineales (operaciones matemáticas). En este paso de la red, es fácil que la información fluya a través de la red sin apenas cambios.

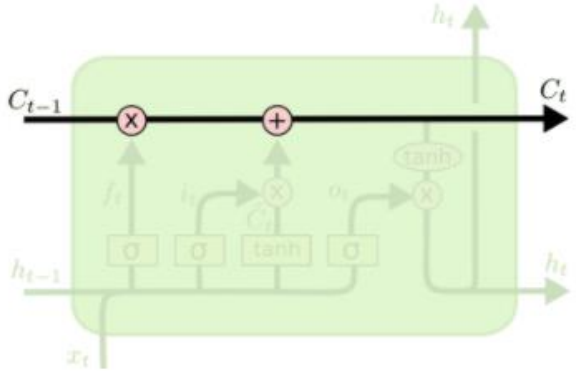


Figura 25 : Estado del módulo. Fuente: Referencia 22.

También, es importante conocer otro elemento del módulo como son las llamadas "*gates*" o "puertas". Estas estructuras tienen la capacidad de regular la agregación o eliminación de información al estado del módulo, y es que estas "puertas" dejan pasar la información de forma opcional. Están compuestas por una capa de red neuronal sigmoidea y una operación de multiplicación.

La capa sigmoidea genera números entre 0 y 1, que cuantifica la información que debe dejarse pasar (0 no deja pasar nada y 1 deja pasar toda la información).

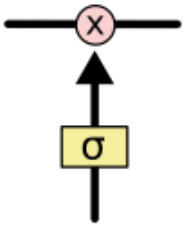


Figura 26 : "Gate" o "puerta" de red LSTM. Fuente: Referencia 22.

En concreto, las puertas que están presentes en el bloque son:

- **Puerta 1:** Indica la eliminación o no de la información almacenada en el estado anterior. También se suele representar con una F (*Forget*).

- **Puerta 2:** Indica si debemos o no almacenar la información de entrada y la predicción anterior. También se suele representar con una M (*Memory*).

- **Puerta 3:** Indica si podemos pasar o no la información a la salida. También se suele representar con una O (*Output*).

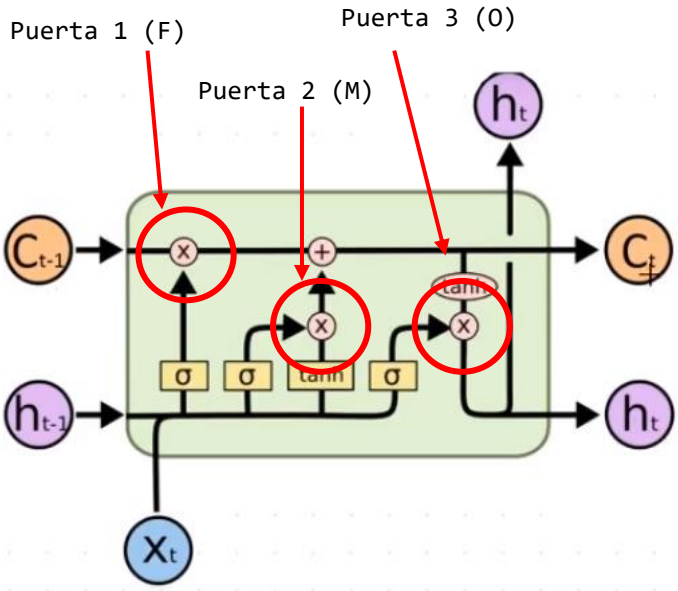
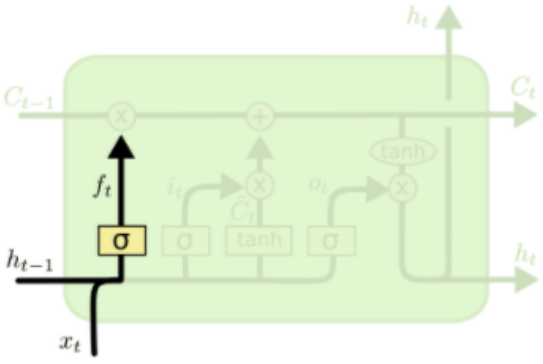


Figura 27: Puertas de red LSTM. Fuente: Referencia 22

Una vez conocidas las dos partes principales de un módulo de red LSTM, vamos a analizar en detalle el funcionamiento de un módulo:

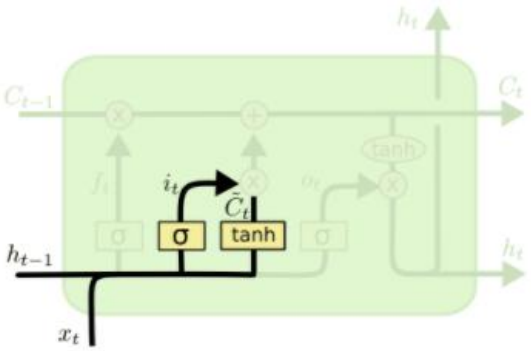
1) El primer paso es decidir qué información se va a eliminar del estado del módulo, esta decisión se lleva a cabo en la capa sigmoidea. Esta capa tiene como entradas h_{t-1} y X_t y genera una salida entre 0 y 1 para cada estado del módulo C_{t-1}. Con esto conseguimos borrar información de estados anteriores que no tienen apenas relevancia en el estado actual, o lo contrario, mantener toda la información de estados anteriores que son relevantes en el estado actual.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 28 : Primer paso en red LSTM. Fuente: Referencia 22.

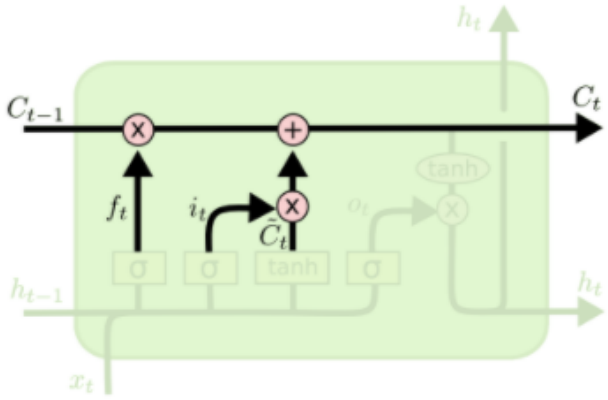
2) El segundo paso es decidir qué información almacenaremos en el estado del módulo, primero a partir de una capa sigmoidea que decide los valores que se van a actualizar y después mediante una capa *tanh* que crea un vector con los nuevos valores, $\hat{C}_t$, que se podrían agregar al valor del estado.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 29 : Segundo paso en red LSTM. Fuente: Referencia 22.

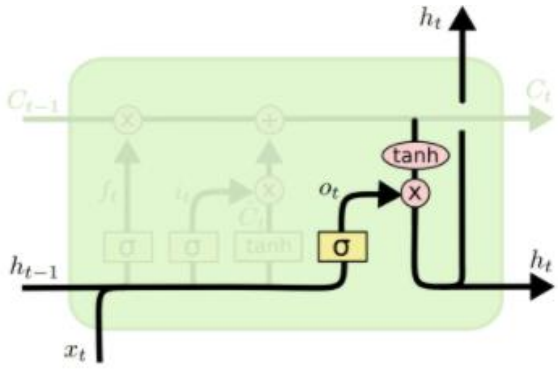
3) El tercer paso es la actualización del estado del módulo anterior C_{t-1}, al nuevo estado C_t. Para ello, multiplicamos el estado anterior por f_t. A continuación, agregamos i_t x $\hat{C}_t$. Estos son los nuevos valores escalados por cuánto decidimos actualizar cada valor de estado.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 30 : Tercer paso en red LSTM. Fuente: Referencia 22.

4) Por último, el cuarto paso es la decisión de lo que se va a agregar a la salida del módulo (h_t). Esta salida estará basada en el estado del módulo pero será una salida filtrada. Primero se pasa por una capa sigmoidea que decide las partes del estado del módulo que se van a generar, luego se pasa el estado por *tanh* que genera valores entre -1 y 1 y lo multiplica por la salida de la "puerta" sigmoidea, de modo que solo producimos las partes que decidimos.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 31 : Cuarto paso en red LSTM. Fuente: Referencia 22.

En resumen, podríamos decir que las redes neuronales LSTM siguen una serie de pasos para decidir la información con la que va a trabajar, que son:

- **Información Eliminada**: gracias a la capa sigmoidea, podemos despreciar información que no tiene importancia en la salida de nuestro modelo. Por ejemplo en un modelo de reconocimiento de texto, podríamos descartar ciertas palabras en base a si anteriormente hay un sujeto masculino o femenino ya que esto nos dará información del tipo de palabra que debe presentarse a la salida.

- **Información Almacenada**: en este paso, primero la capa sigmoidea decide los valores que se van a actualizar para después generar un vector con los valores que podrían conformar la salida del estado. Siguiendo el ejemplo del texto, podríamos decidir almacenar un nuevo sujeto que reemplace a uno anterior que ya no tiene relevancia.

- **Actualización del estado:** Multiplicamos el estado anterior por el resultado de la primera capa sigmoidea y a esto se le suma el resultado de multiplicar la salida de la capa sigmoidea con la capa *tanh*. Con esto conseguimos obtener las nuevas posibles salidas escaladas según el valor resultante de actualizar el estado del bloque. En nuestro ejemplo, este paso elimina la idea del género del sujeto almacenada y añadiría la nueva.

- **Decisión de salida:** En el último paso, gracias a la capa sigmoidea y a la *tanh*, se decide las partes del nuevo estado que serán parte de la salida. Lo que en el ejemplo significaría que después de obtener el nuevo sujeto, obtendríamos información del mismo para la siguiente palabra, como puede ser si se trata de un sustantivo singular o plural.

4.2.3 Análisis de datos

Una vez que hemos estudiado el modelo de red neuronal que queremos aplicar, vamos a hacer un análisis y un preprocesamiento de los datos con los que vamos a desarrollar nuestro modelo.

Partimos de la base de datos del ECDC, y adaptamos la base de datos a un formato con el que podamos trabajar con mayor facilidad, eliminamos la columnas que no tienen ninguna relevancia en este estudio y clasificamos la columna de los grupos de edad y de la fecha con valores numéricos secuenciales, es decir, en la columna "*age_group*" redefinimos los valores con el siguiente criterio:

Antes	Ahora
<15yr	0
15-24yr	1
25-49yr	2
50-64yr	3
65-79yr	4
80+yr	5

Tabla 1: Clasificación "age_*group*"

Esto lo cambiamos con la ayuda de la función de pandas "*apply*", que nos ayuda a sustituir la cadena de texto por los valores numéricos que asignamos:

```
# Grupo 0: <15yr
df["age_group"]=df["age_group"].apply(lambda x: 0 if x=="<15yr" else x)
# Grupo 1: 15-24yr
df["age_group"]=df["age_group"].apply(lambda x: 1 if x=="15-24yr" else x)
# Grupo 2: 25-49yr
df["age_group"]=df["age_group"].apply(lambda x: 2 if x=="25-49yr" else x)
# Grupo 3: 50-64yr
df["age_group"]=df["age_group"].apply(lambda x: 3 if x=="50-64yr" else x)
# Grupo 4: 65-79yr
df["age_group"]=df["age_group"].apply(lambda x: 4 if x=="65-79yr" else x)
# Grupo 5: 80+yr
df["age group"]=df["age group"].apply(lambda x: 5 if x=="80+yr" else x)
```

Realizamos un proceso similar a la columna que contiene la fecha (dada con el siguiente formato: año - semana del año) de los datos "*year_week*":

Antes	Ahora
2020-05	0
2020-06	1
2020-07	2
...	...
2021-19	67
2021-20	68

Tabla 2: Clasificación "*year_week*"

Al igual que con la columna de "*age_group*", en la de "*year_week*" aplicamos la función de pandas "*apply*":

```
# Semana 0: 2020-05
df["year_week"]=df["year_week"].apply(lambda x: 0 if x=="2020-05" else x)
# Semana 1: 2020-06
df["year_week"]=df["year_week"].apply(lambda x: 1 if x=="2020-06" else x)
# Semana 2: 2020-07
df["year_week"]=df["year_week"].apply(lambda x: 2 if x=="2020-07" else x)
...
...
# Semana 67: 2021-19
df["year_week"]=df["year_week"].apply(lambda x: 67 if x=="2021-19"else x)
# Semana 68: 2021-20
df["year_week"]=df["year_week"].apply(lambda x: 68 if x=="2021-20"else x)
```

Con estas modificaciones, tenemos la base de datos con el siguiente formato:

	country	country_code	year_week	age_group	new_cases	population	rate_14_day_per_100k
6745	Malta	MT	16	2	25	205154	22.9
3497	Germany	DE	55	5	4537	5681135	172.3
7115	Netherlands	NL	64	1	10557	2143743	952.9
952	Croatia	HR	59	2	3798	1319478	459.8
2235	Denmark	DK	18	5	11	272326	12.5
5303	Italy	IT	48	3	27961	13307545	379.1
10386	Sweden	SE	9	4	673	1529061	77.2
2649	Estonia	EE	47	5	242	77445	569.4
4342	Iceland	IS	61	0	12	68219	51.3

Figura 32: Muestra de preprocesamiento de base de datos. Fuente: Elaboración propia.

4.2.4 Representación de datos

En vista a la gran cantidad de datos de los que disponemos, vamos a centrar el estudio en los datos de España, ya que lo que buscamos es poder sintetizar una conclusión coherente. Y al finalizar el proyecto realizaremos un estudio comparativo de los países seleccionados previamente, por lo que el resto de datos no los utilizaremos.

Comenzamos con una representación para tener una referencia de los datos con los que vamos a trabajar y los cuales queremos predecir, en nuestro caso centraremos el estudio en los datos de nuevos casos de COVID por edades para poder hacer una predicción de los futuros brotes que pueda haber [24]:

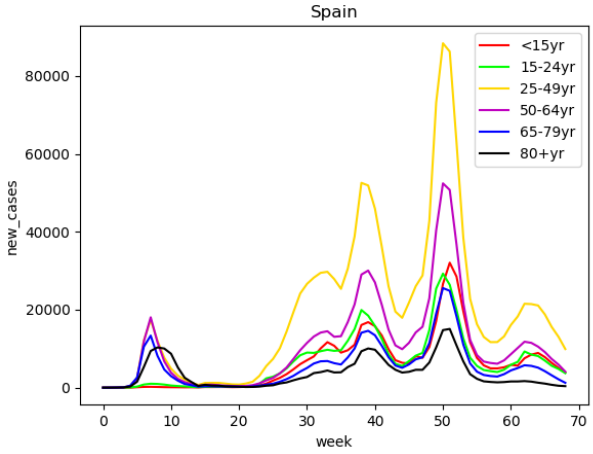


Figura 33 : Casos de COVID por rangos de edad. Fuente: Elaboración propia.

Visualizamos la evolución de los casos de COVID en España de todos los rangos de edad y desde la semana 0 (5ª semana del 2020) hasta la semana 68 (20ª semana del 2021).

En esta representación podríamos concluir que la población más afectada por la enfermedad es la del rango de edad 25-49 años y la menos afectada las personas mayores de 80 años, pero esto sería una conclusión errónea ya que en esta gráfica no estamos teniendo en cuenta una tasa sino unos valores absolutos.

La pirámide de población del país nos muestra la cantidad de personas que forman cada rango de edad. Para generarla, hacemos uso de la columna "*population*" de nuestra base de datos.

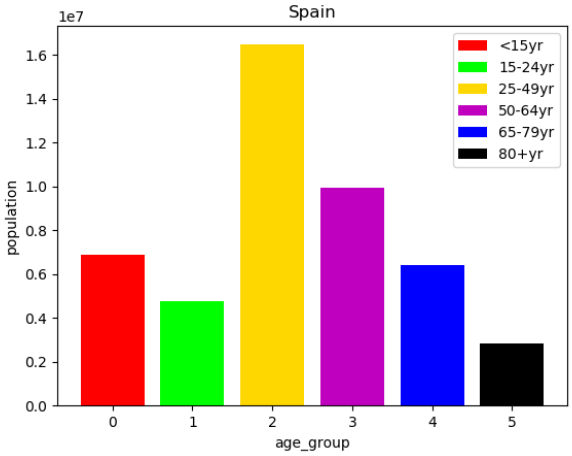


Figura 34 : Pirámide de poblacion España. Fuente: Elaboración propia.

Con esto demostramos que la elevada cifra de contagiados por COVID en el rango de edad de 25-49 años se debe a la gran cantidad de población existente.

Una vez estudiado los datos de los que disponemos y su distribución, vamos a intentar crear una red neuronal LSTM para predecir los nuevos casos que se darán de la enfermedad para todos los rangos de población.

4.2.5 Aplicación de modelo

Para aplicar el modelo, vamos a segregar la información de acuerdo a los grupos de edad, así pues, comenzamos la aplicación del modelo con el grupo 0 que comprende la población menor de 15 años. Para el resto de los grupos de edad, podemos encontrar las gráficas asociadas en el Anexo A. [25][26]

4.2.5.1 Selección de muestras para entrenamiento y test.

Nuestro objetivo es entrenar la red LSTM con los datos de las primeras 55 semanas (*data[:54]*) y comprobar su funcionamiento (precisión) con los datos de las últimas 14 semanas (*data[55:]*).

```
# Set de datos de entrenamiento y test.
set_train = data[:54]
set_test = data[55:]

Número de muestras:      69
Muestras entrenamiento: 55
Muestras test:           14
```

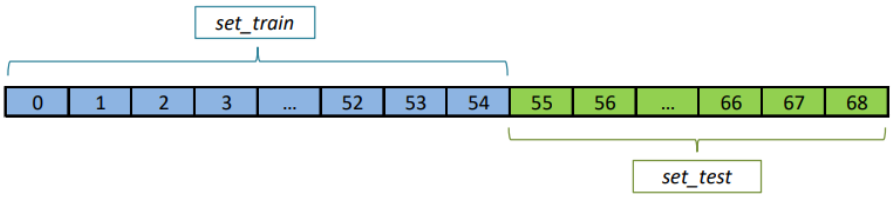


Figura 35: *set_train* y *set_test*. Fuente: Elaboración propia.

A continuación, representamos la idea principal de la repartición de datos para generar el modelo.

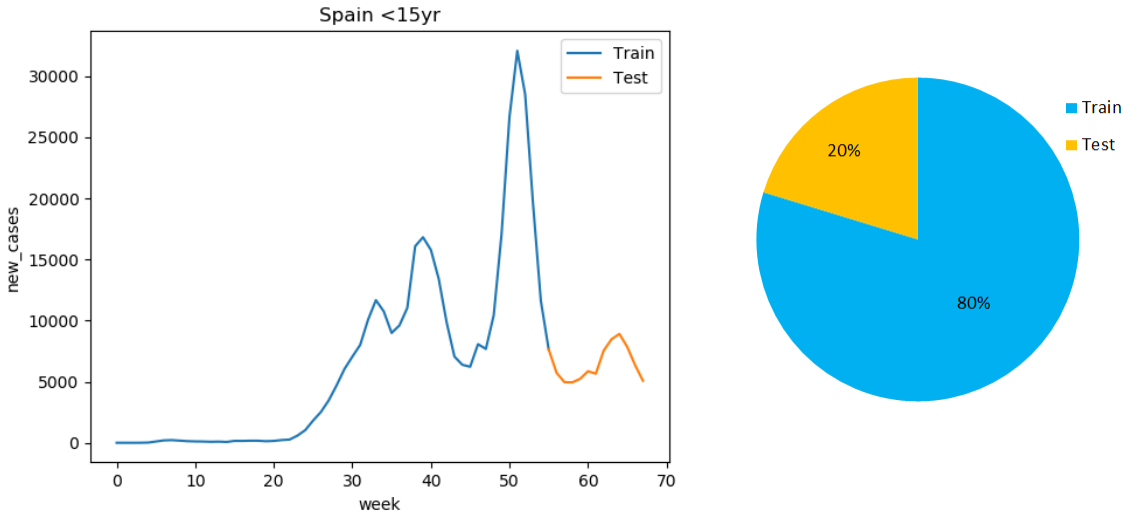


Figura 36 : Datos "Train" y "Test". Fuente: Elaboración propia.

4.2.5.2 Normalización de datos.

Normalizamos los valores de entrenamiento ("*set_train*") con la función "*MinMaxScaler*" pasando el parámetro "*feature_range*" el cuál especifica que queremos normalizar los valores a una escala entre 0 y 1, y guardamos los nuevos valores normalizados en otra variable, en este caso "*set_train_scale*". Aplicamos normalización y no estandarización porque tenemos una red neuronal recurrente, con función sigmoidea en la capa de salida y buscamos tener datos entre 0 y 1.

```
# Normalización de los datos
sc = MinMaxScaler(feature_range=(0,1))
set_train_scale = sc.fit_transform(set_train)
```

Para poder aplicar la normalización a nuestros datos de entrenamiento, debemos usar *fit_transform* que aplica a las muestras *set_train* el escalado de los datos, y lo guardamos en *set_train_scale*.

Para ver la normalización, representamos los datos para contemplar las escalas:

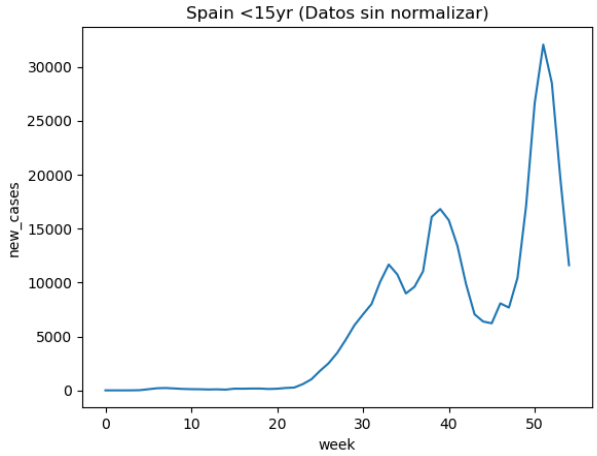


Figura 37 : Datos sin normalizar ("set_train"). Fuente: Elaboración propia.

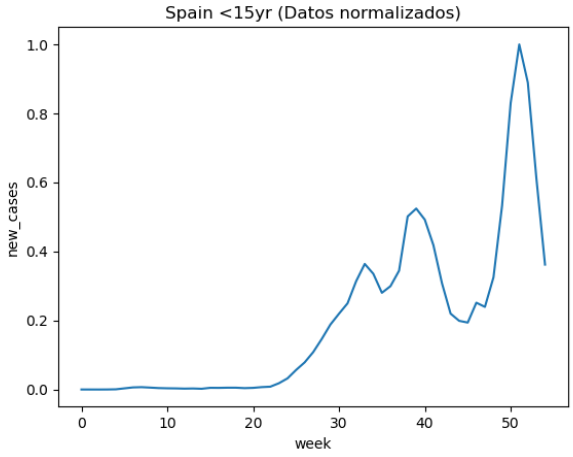


Figura 38 : Datos normalizados ("set_train_scale"). Fuente: Elaboración propia.

4.2.5.3 Datos entrada red neuronal.

Una vez tenemos los datos normalizados entre 0 y 1, preparamos los datos que vamos a introducir en la red neuronal.

Nuestra red neuronal va a ser entrenada con bloques de datos (pasos temporales). Con la variable "*time_step*" especificamos que el tamaño de los bloques será 2, y es que para cada dos datos queremos que nos genere una salida o dicho de otra manera, nuestro modelo va a analizar los dos valores de casos de COVID anteriores y basándose en estos valores va a generar una salida o predicción. Hay que resaltar que dos valores contemplan un rango temporal de dos semanas para hacer la predicción de la siguiente.

Creamos dos listas "*X_train*" e "*Y_train*", en las que iremos almacenando (mediante un bucle que itera tantas veces como número de valores tenemos de entrenamiento) los valores que toma la entrada y salida de la red para los datos de entrenamiento.

```
time_step = 2
X_train = []
Y_train = []

for i in range(0,len(set_train_scale)-time_step):
    X_train.append(set_train_scale[i:i+time_step,0])
    Y_train.append(set_train_scale[i+time_step,0])
```

X_train será una lista en la que iremos almacenando los datos en conjuntos (*arrays*) del tamaño *time_step*, en este caso 2 y en *Y_train* se almacenará la predicción para cada conjunto (que será el siguiente dato del conjunto de entrenamiento).

Recordamos que nuestro conjunto de datos de entrenamiento va desde el dato 0 hasta el dato 54, por lo que nuestro bucle *for* se moverá en el rango (0,53). Y presentamos brevemente cómo será el contenido de las listas:

X_train = [(Dato 0, Dato 1), (Dato 1, Dato 2) … (Dato 51, Dato 52),(Dato 52, Dato 53)]
Y_train = [Dato 2, Dato 3… Dato 53, Dato 54]

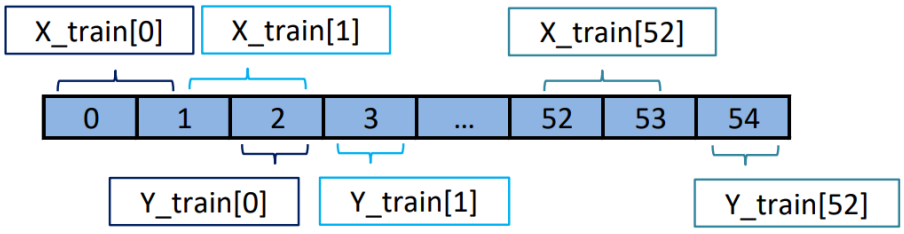


Figura 39: *X_train* e *Y_train*. Fuente: Elaboración propia.

Después, convertimos las listas resultantes a "*numpy arrays*" ya que por temas de optimización nuestro código nos lo requiere.

```
X_train, Y_train = np.array(X_train), np.array(Y_train)
```

Luego, vamos a transformar "*X_train"* (actualmente de dos dimensiones) a tres dimensiones con la función "*reshape*" por ser un requisito de la librería "*keras".*

El primer argumento que se le pasa a "reshape" es la matriz que contiene los datos, y el segundo argumento estará compuesto de tres parámetros (*batch_size:* número de muestras total, *timesteps:* número de columnas que tiene *X_train*, que es igual a la variable declarada como *time_step, input_dim:* dimensión que se va a agregar)

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],1))
```

Finalmente, pasaremos a nuestra red neuronal los siguientes parámetros:

```
# Dimensión de entrada
dim_entrada = (X_train.shape[1],1)

# Dimensión de salida
dim_salida = 1

# Número de neuronas
na = 20
```

4.2.5.4 Red neuronal.

Inicializamos el modelo con la función "*Sequential*" de la librería "*keras*", y es que la estructura de datos principal en "*keras*" es la clase "*Sequential*", que permite la creación de una red neuronal con la librería "*Sequential class*", que es un contenedor para el modelo de red secuencial que ofrece "*keras*".

Luego, añadimos las capas de la red neuronal. El modelo "*keras*" se considera como una secuencia de capas y cada uno de ellas "filtra" gradualmente los datos de entrada para obtener la salida deseada. En este modelo podemos encontrar todos los tipos de capas necesarios que se pueden agregar fácilmente a través del método "*add*".

En primer lugar declaramos la capa LSTM a la que le pasamos los siguientes parámetros: número de neuronas de la capa ("*units*") y la dimensión de entrada ("*input_shape*"). Por último, declaramos la capa de salida que será de tipo "*Dense*" a la que se le pasa la dimensión de salida.

Podemos comprobar la arquitectura del modelo con la función "*summary*".

```
red = Sequential()
red.add(LSTM(units=na, input_shape=dim_entrada))
red.add(Dense(units=dim_salida))
red.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 5)                 140
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 6
=================================================================
Total params: 146
Trainable params: 146
Non-trainable params: 0
_________________________________________________________________
```

Gracias a la función "*summary*" podemos ver que se requieren de 146 parámetros que se corresponden a la suma de los 140 parámetros de la primera capa y 6 de la última.
El número de parámetros de la red LSTM sigue la siguiente fórmula:

$4*((dim\_input+1)*dim\_output+dim\_output^2)$

$4*((1+1)*5+5^2)=4*(2*5+25)=4*35=140$

Donde,

dim_input: tamaño de datos de entrada (se le suma 1 de sesgo)
dim_output: viene dado por el número de neuronas

El siguiente paso en la creación de la red neuronal es la configuración del aprendizaje. Podemos configurar cómo será el proceso de aprendizaje con el método "*compile*", con el que podemos especificar algunas propiedades a través de los argumentos del método.

```
red.compile(optimizer='rmsprop', loss='mse')
```

El primer argumento es el optimizador que es la forma en que tenemos que especificar el algoritmo de optimización que permite a la red neuronal calcular los pesos de los parámetros a partir de los datos de entrada y la función de pérdida definida, en nuestro caso elegimos el optimizador "*rmsprop*" que mantiene un promedio del cuadrado de gradientes y divide el gradiente por la raíz de ese promedio, además es el optimizador recomendado para realizar redes recurrentes. Y como último argumento, definimos la función de pérdidas que usaremos para evaluar el grado de error entre las salidas calculadas y las salidas deseadas de los datos de entrenamiento, en nuestro caso elegimos "*mse (mean squared error)*" que calcula la media de cuadrados de errores entre los resultados reales y las predicciones.

Una vez que nuestro modelo ha sido definido y el método de aprendizaje configurado, está listo para ser entrenado. Para esto podemos entrenar o "ajustar" el modelo a los datos de entrenamiento disponibles invocando al método "*fit*" del modelo.

```
red.fit(X_train, Y_train, epochs=40,batch_size=5)
```

En los dos primeros argumentos hemos indicado los datos con los que entrenaremos el modelo ("*X_train*" e "*Y_train*").

El argumento "*batch_size*" indica el número de datos que usaremos para cada actualización de los parámetros del modelo (número de "lotes" en el que queremos que nuestros datos se analicen para después cambiar el valor de los pesos) y con las "*epochs*" indicaremos el número de veces que usaremos todos los datos en el proceso de aprendizaje (número de iteraciones con el que queremos entrenar el modelo).

Este método encuentra el valor de los parámetros de la red a través de cada iteración de este algoritmo, este toma los datos de entrenamiento de "*X_train*", los pasa a través de la red neuronal (con los valores que tienen sus parámetros en ese momento), compara el resultado obtenido con el esperado (indicado en "*Y_train*") y calcula el error para guiar el proceso de ajuste de los parámetros del modelo, que consiste intuitivamente en aplicar el optimizador especificado anteriormente para calcular un nuevo valor de cada uno de los parámetros del modelo en cada iteración, de tal manera que se va reduciendo el error.

Este es el método que, como veremos, puede llevar más tiempo y "*keras*" nos permite visualizar su progreso utilizando el argumento detallado (por defecto, igual a 1), además de indicar una estimación de cuánto tiempo toma cada época.

```
Epoch 1/40
11/11 [==============================] - 2s 3ms/step - loss: 0.0434
Epoch 2/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0270
Epoch 3/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0387
Epoch 4/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0219
Epoch 5/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0256
…
…
…
Epoch 35/40
11/11 [==============================] - 0s 3ms/step - loss: 0.0137
Epoch 36/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0094
Epoch 37/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 38/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0217
Epoch 39/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0098
Epoch 40/40
11/11 [==============================] - 0s 2ms/step - loss: 0.0176
```

Aquí, podemos observar que el valor de "*loss*" (la función de pérdidas), mejora en cada época gracias al optimizador que hemos empleado. Por lo que podemos concluir que nuestro modelo está aprendiendo.

Y por último, creamos una variable donde estén todos los valores necesarios para predecir las muestras de test, esto incluye añadir tantos datos del conjunto de entrenamiento como sea el tamaño de bloque seleccionado (en nuestro caso es 2). Así, podremos predecir el primer valor del conjunto de test.
Recordamos que nuestro conjunto de test comprende los datos de las semanas 55 hasta la 68. Por lo que si queremos predecir los últimos 14 valores de nuestro conjunto de datos, debemos recuperar del conjunto de entrenamiento los valores del tamaño del bloque *time_step*, y asi podremos predecir el valor del Dato 55 y del Dato 56.

```
dataset = pd.concat((set_train,set_test), axis=0)
auxTest = dataset[len(dataset)-len(set_test)-time_step:].values
```

auxTest = [Dato 53, Dato 54, Dato 55, Dato 56… Dato 66, Dato 67, Dato 68]

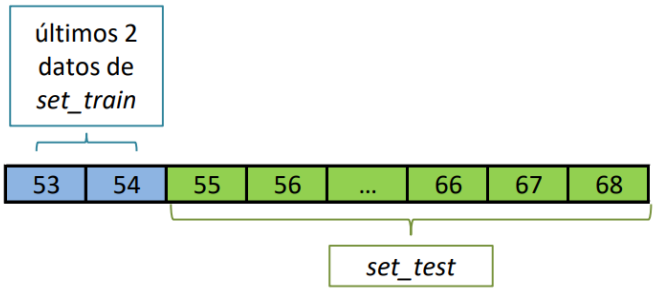


Figura 40: Conjunto datos test. Fuente: Elaboración propia.

Como hemos concatenado los conjuntos de datos originales (*set_train* y *set_test*), debemos aplicar un escalado a los datos, ya que nuestra red ha sido entrenada con datos normalizados y queremos mantener la consistencia del modelo. Pero primero debemos transformar nuestro conjunto de datos para tener un correcto formato de *array*, esto lo conseguimos gracias a la función "*reshape*", que mantiene los datos que tenemos pero formando un *array* bien dimensionado.

```
auxTest = auxTest.reshape(-1,1)
auxTest = sc.transform(auxTest)
```

Esta vez hemos utilizado *transform* en vez de *fit_transform*, esto se debe a que *fit_transform* aplica una normalización en base a la media y la varianza de los datos, por lo que para mantener ese criterio debemos utilizar esas características que *transform* nos puede proporcionar ahora.

Ahora, al igual que hicimos en entrenamiento, realizamos un bucle que recorra todos los valores de los datos de test, convertimos *X_test* a *nunmpy array* y redimensionamos:

```
X_test = []
for i in range(0,len(auxTest)- time_step):
    X_test.append(auxTest[i:i+time_step,0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
```

Con esto, hacemos la predicción y desescalamos la predicción para tener los valores desnormalizados.

```
prediccion = red.predict(X_test)
prediccion = sc.inverse_transform(prediccion)
```

Finalmente, los resultados obtenidos son:

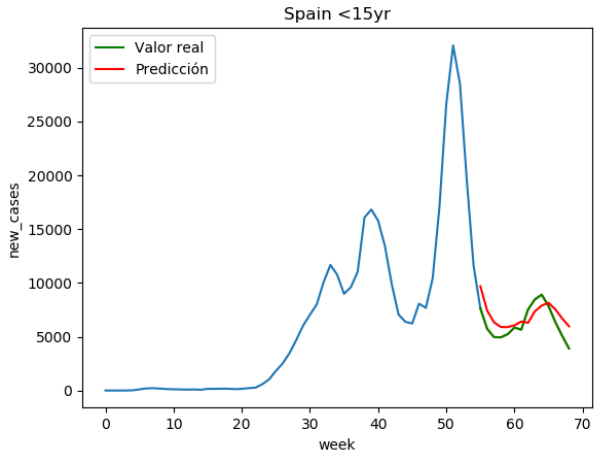


Figura 41 : Predicción vs Valor real. Fuente: Elaboración propia.

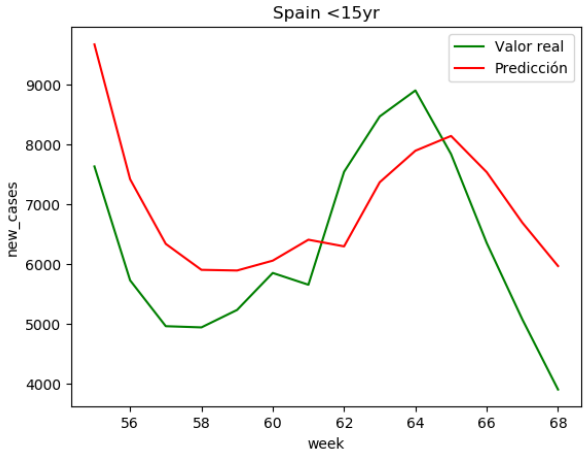


Figura 42: Ventana Predicción vs Valor real. Fuente: Elaboración propia.

Observamos que la predicción es incapaz de replicar picos abruptos, es decir el modelo no es lo suficientemente exacto como para predecir irregularidades no lineales.
Y es que para este tipo de datos no podemos obtener resultados exactos, sino que debemos valorar una tendencia (podemos considerar que los resultados son buenos ya que se mueven en un mismo intervalo de número de casos y con un comportamiento parecido).

Con datos de enfermedades no podremos conseguir una alta precisión, ya que al igual que por ejemplo con los datos de la bolsa de valores; estos datos son dependientes de muchas variables y no presentan un comportamiento que pueda predecirse con exactitud.

A la hora de evaluar matemáticamente el rendimiento de la red, calculamos el error medio en los resultados, calculando la diferencia presente entre las salidas de la red (predicción) y los valores reales (set_test):

```
error_medio = abs(prediccion-set_test).mean()
```

Grupo de edad	Error	Porcentaje de error
<15 años	1224	13.6%
15-24 años	1261	14%
25-49 años	2326	10.6%
50-64 años	1461	12.7%
65-79 años	885	16%
>80años	562	23%

Tabla 3: Error del modelo. Fuente: Elaboración propia.

El error es el número de casos medio que se aleja la predicción de la realidad. Por lo que hacemos un cálculo del porcentaje de error que esto supone y obtenemos que el mayor error se da en los resultados del grupo de edad de mayores de 80 años, como podemos comprobar en la figura 57 del Anexo A.

5 Integración, pruebas y resultados

5.1 *Integración y pruebas*

En este capítulo, se mostrarán los resultados obtenidos de las pruebas realizadas para optimizar los parámetros de la red neuronal.

En primer lugar, hemos decidido establecer el número de muestras total (69) en 80% muestras para entrenamiento (55) y 20% para test (14).

A continuación, probamos a entrenar nuestra red con un conjunto de entrenamiento más pequeño y un conjunto de test mayor. En concreto entrenaremos la red con las 45 primeras muestras y las probaremos con las últimas 24 muestras de nuestro conjunto de datos.

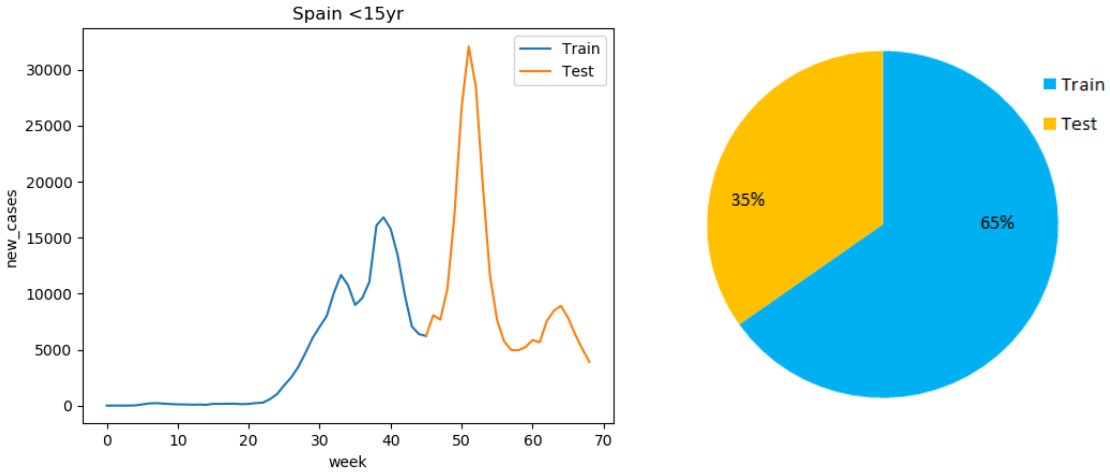


Figura 43: Reparto de datos para prueba de Red LSTM. Fuente: Elaboración propia.

Si entrenamos a la red con menos muestras los resultados son:

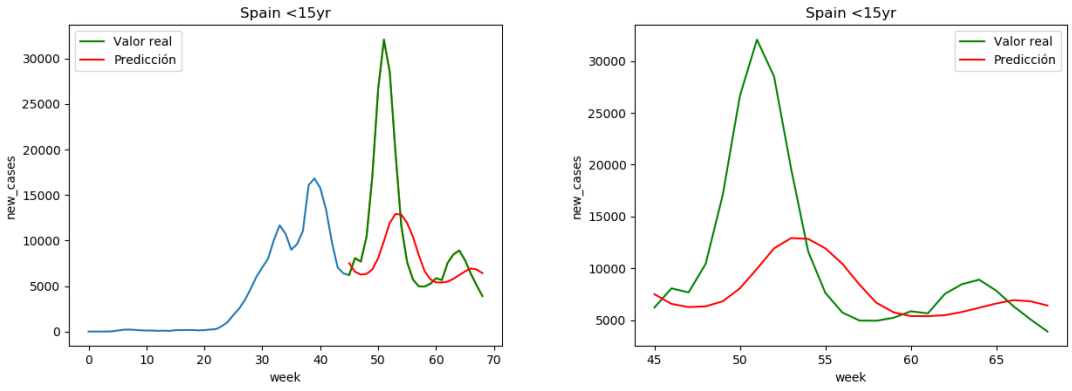


Figura 44 : Prueba con muestras: 65% train, 35% test. Fuente: Elaboración propia.

Al entrenar la red con menos muestras, nuestro modelo es incapaz de predecir las curvas tan abruptas. Predice ligeramente la tendencia aunque como se puede ver, la desigualdad presente entre los valores reales y los predichos es notablemente alta.

Una vez hemos establecido el número de muestras de cada tipo con el que trabajamos, decidimos probar diferentes parámetros de la red.

Los cuatro parámetros con los que realizamos pruebas son:

- *time_step*
- Número de neuronas
- *epochs*
- *batch_size*

Recordamos que el valor de *time_step* es una variable que nos dice los pasos temporales que vamos a tener en la entrada de la red, por lo que si aumentamos su tamaño formaremos menos grupos de entrada de la red. En concreto, tendremos a la entrada de la red los siguientes números de conjuntos de datos: Nº total de muestras - *time*_step + 1.

Por ejemplo, si tuviéramos una base de datos con 10 muestras, obtendríamos los siguientes números de conjuntos de datos a la entrada de la red:

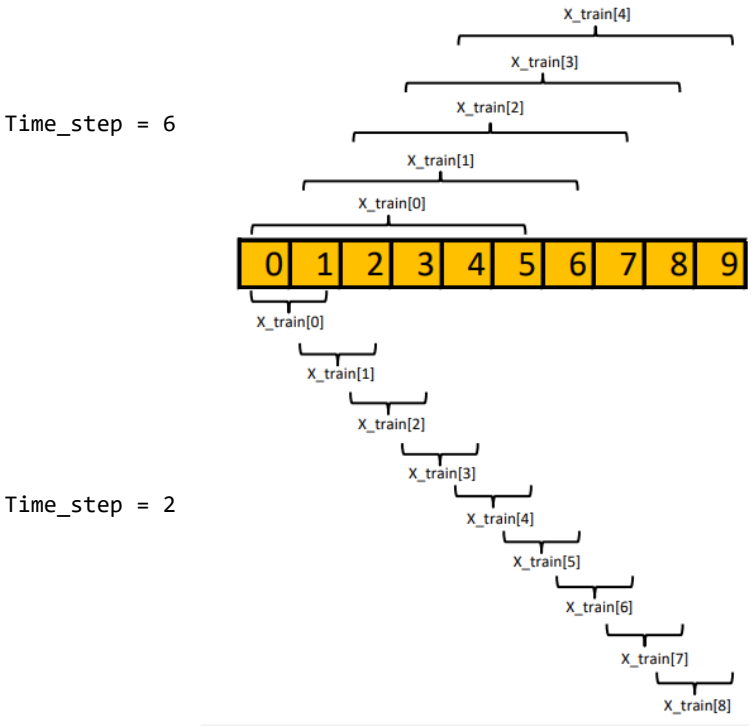


Figura 45: Diferentes *time_step*. Fuente: Elaboración propia.

Con esto comprobamos que al tener un *time_step* más pequeño, podremos introducir mayor conjunto de datos a la entrada de la red.

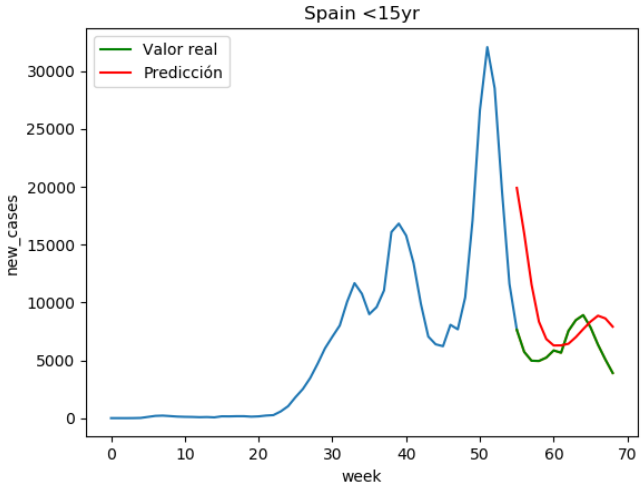


Figura 46: Prueba con *time_step* = 6. Fuente: Elaboración propia.

Probamos con *time_step* = 6, y vemos que los resultados son malos por lo que decidimos reducir el valor a 2 en base a los resultados obtenidos.

Por otro lado, buscamos entre los parámetros que son decisivos para el rendimiento de la red, el número de *epochs* y de neuronas. La red neuronal es mucho más rápida cuanto menos *epochs* y neuronas tenga, así que decidimos probar con 2 neuromas y 20 epochs.

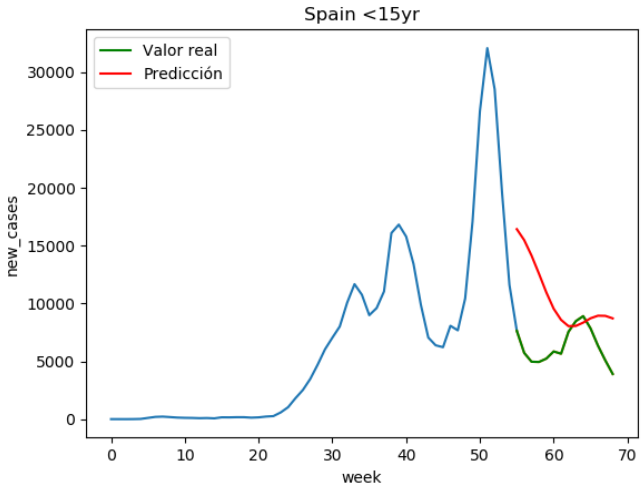


Figura 47 : Prueba con 2 neuronas y 20 epochs. Fuente: Elaboración propia.

Como podemos ver tanto con el número de neuronas como *epochs*, al disminuir su valor, los resultados se degradan notablemente. Por lo que declaramos como valores óptimos 5 neuronas y 40 *epochs*. Podríamos utilizar valores mayores pero no tendríamos una buena relación entre rendimiento y resultados, es decir, tendríamos resultados similares con un mayor tiempo de computación.

En el caso del *batch_size*, al aumentar su valor, las predicciones empeoran por lo que debemos analizar los datos en lotes de 5 para que después la red pueda actualizar los pesos en base a este análisis. Para demostrar esto, hacemos una prueba con un valor de *batch_size* de 20.

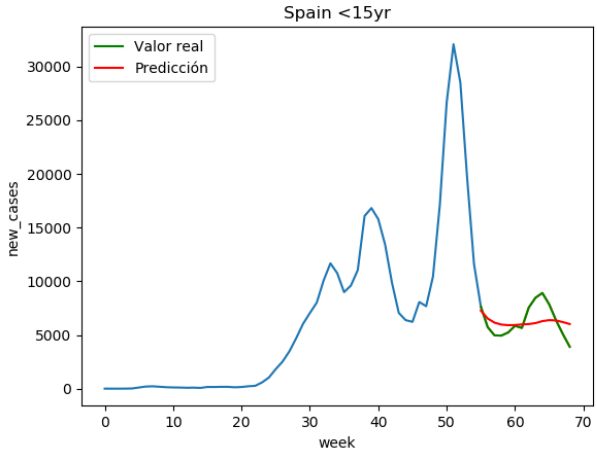


Figura 48 : Prueba con mayor número de batch_size (20). Fuente: Elaboración propia.

Además de hacer un análisis de los resultados de las pruebas con diferentes parámetros, también analizamos el comportamiento del modelo con más capas. Añadimos dos capas LSTM a la existente:

```
red = Sequential()
red.add(LSTM(units=na, return_sequences=True, input_shape=dim_entrada))
red.add(LSTM(units=na, return_sequences=True))
red.add(LSTM(units=na, return_sequences=True))
red.add(LSTM(units=na))
red.add(Dense(units=dim_salida))
red.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_10 (LSTM)               (None, 2, 5)              140
_________________________________________________________________
lstm_11 (LSTM)               (None, 2, 5)              220
_________________________________________________________________
lstm_12 (LSTM)               (None, 2, 5)              220
_________________________________________________________________
lstm_13 (LSTM)               (None, 5)                 220
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 6
=================================================================
Total params: 806
Trainable params: 806
Non-trainable params: 0
_________________________________________________________________
```

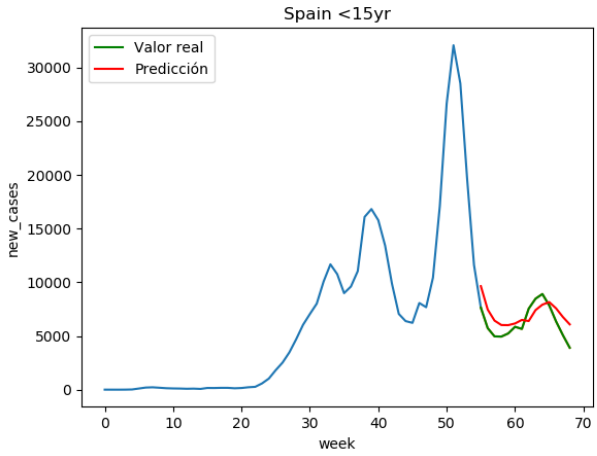


Figura 49: Prueba añadiendo 2 capas LSTM. Fuente: Elaboración propia.

Observamos que al añadir más capas obtenemos resultados parecidos y no es compensable el tiempo de computación que dedica el modelo, el cual aumenta notablemente.

Finalmente, los valores donde consideramos que el tiempo de computación y los resultados obtenidos son los adecuados son:

- *time_step:* 2
- Número de neuronas: 5
- *epochs:* 40
- *batch_size:* 5
- Capas de la red neuronal: 2 (1 capa LSTM y 1 capa *Dense*)

También hemos realizado pruebas con diferentes optimizadores como *Adam* (descenso por gradiente estocástico), pero observamos que en el entorno de pruebas, siempre arrojaban mejores resultados aquellas redes con optimizador *rmsprop*.

Para las pruebas realizadas correspondientes al grupo de edad menor de 15 años, obtenemos los siguientes porcentajes de error:

Prueba	Error	Porcentaje de error
Figura 44	4910	54.5%
Figura 46	4404	48.9%
Figura 47	4385	48.7%
Figura 48	3601	40.1%
Figura 49	1102	12.2%

Tabla 4: Error pruebas del modelo. Fuente: Elaboración propia.

El modelo con 2 capas LSTM presenta tan solo un 1.4% menos de error respecto al modelo implementado con 1 capa LSTM.

5.2 *Resultados*

Los datos obtenidos de aplicar la red neuronal nos dicen que es posible evaluar la evolución de la enfermedad, al menos predecir una tendencia. Este ejercicio nos sirve para sintetizar ideas de manera global, por lo que vamos a ayudarnos de otro tipo de información para poder generar una deducción coherente.

En nuestro caso, vamos a estudiar los datos de vacunación; en concreto, vamos a analizar la estrategia de vacunación que han llevado a cabo los países que son objeto de estudio bajo este proyecto: España, Luxemburgo, Croacia y Letonia. Como hemos visto en el análisis de la base de datos de WB son los países que consideramos nos pueden aportar información de la influencia de los factores económicos y ambientales o genéticos (que son los factores que pueden construir las bases de la esperanza de vida).

Como podemos ver en el Anexo B, la evolución de los casos en España y Luxemburgo ha tenido la misma tendencia (podríamos decir que los casos se han incrementado y disminuido en "olas"; podemos observar que ha habido 4 "olas"), por otro lado Croacia y Letonia han tenido una tendencia incremental a mediados de la escala temporal observada, es decir, al principio apenas ha tenido impacto la enfermedad, pero después de 35 semanas, el número de casos de COVID se ha incrementado notablemente y ha dado lugar a 2 "olas".

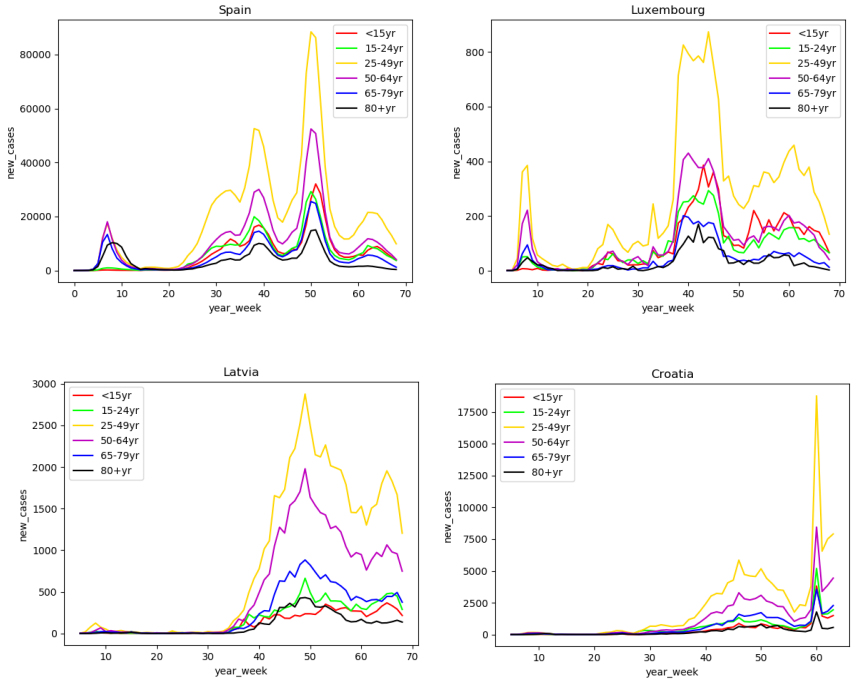


Figura 50 : Gráficas Anexo B. Dos tipos de evoluciones. Fuente: Elaboración propia.

Viendo que hay dos formas de evolución, analizamos el proceso de vacunación.

En los anexos C, D, E y F. Podemos encontrar todas las gráficas de vacunación por grupos de edad que se han producido desde que comenzaron con las inoculaciones (finales Diciembre 2020) hasta mediados de Mayo 2021.

Se observa que al igual que hay dos tipos de evoluciones de los casos de COVID, en las vacunaciones, para España y Letonia tenemos una misma estrategia de vacunación en la que la población de entre 18 y 49 años ha recibido la vacuna desde el principio (posiblemente personas del sector sociosanitario), mientras que en Letonia y Croacia este grupo de población ha sido el último en comenzar su vacunación, cediendo las primeras dosis para el grupo de población mayores de 70 años.

Extrayendo las gráficas de los anexos, podemos observar mejor las tendencias que menciono, por ejemplo en las gráficas del grupo de edad de 18-24 años.

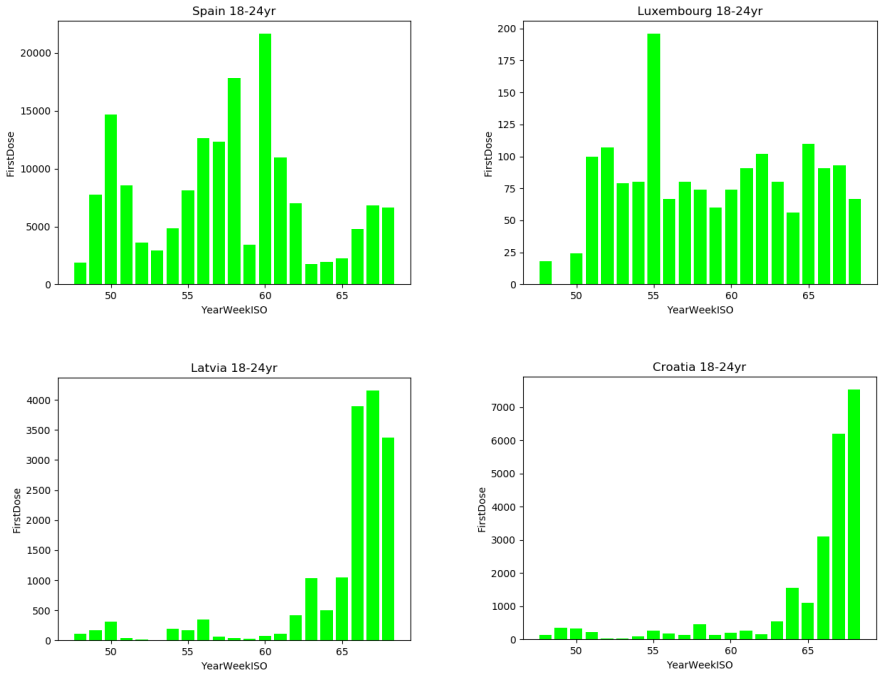


Figura 51 : Gráficas Anexos C, D, E y F. Fuente: Elaboración propia.

Es decir, el país con mayor PIB (Luxemburgo) y el que tiene mayor esperanza de vida (España) ha seguido una estrategia de vacunación similar, al igual que el país con menor PIB (Croacia) y el país con menor esperanza de vida (Letonia).

Finalmente, podemos extraer como conocimiento, que la evolución de las enfermedades globales pueden tener cierta correlación con factores económicos y ambientales.

6 Conclusiones y trabajo futuro

6.1 *Conclusiones*

Con este proyecto hemos podido demostrar que aquellos países que presentan factores del índice de desarrollo superiores (mayor PIB y esperanza de vida), presentan una evolución de la enfermedad y estrategia de vacunación similares, al igual que entre los países con factores de IDH inferiores.

Por otro lado, se trabajaba desde la hipótesis de que a partir de los datos históricos que tuviéramos de la evolución de la pandemia, podríamos hacer una predicción de los futuros datos. Es decir, con los casos de COVID de las primeras semanas desde el inicio de la expansión de la enfermedad, podríamos predecir los casos que habría en un país distinguiendo incluso por rango de edad en las próximas semanas.

Al generar un modelo con el que podemos predecir la evolución de los casos de COVID, hemos podido evaluar que las redes LSTM ayudan a aumentar la calidad de las predicciones. Aunque los resultados de la generación de la red neuronal LSTM han sido bastante buenos, hay que tener en cuenta que modelar los contagios de una pandemia de estas características conlleva que sea difícil predecir los valores, pero sí que podemos estudiar la tendencia.

La propia naturaleza del problema puede ser un factor restrictivo, ya que la información de entrada puede ser insuficiente o no determinista. Y es que predecir con exactitud los contagios que va a haber por el virus tiene muchas variables determinantes que son imposibles de trasladar a una misma base de datos, por lo que tan solo podemos contemplar la tendencia de los datos que sí tenemos reflejados y analizar la existencia de datos desconocidos matemáticamente hablando, pero que sí sabemos que pueden modificar la evolución de la pandemia, como pueden ser los datos de: las medidas tomadas por cada región, los sistemas de Sanidad, las medidas proactivas como las mascarillas, gel hidroalcohólico, distancia de seguridad…

Analizando el trabajo realizado (proyecto de *Machine Learning*) podemos concluir que la elección de los parámetros de optimización, así como las muestras con las que trabajamos es la parte más importante ya que tienen mayor impacto en los resultados.

Conocer las herramientas de M*achine Learning* y los modelos disponibles, es esencial a la hora de generar un algoritmo, ya que debemos centrar el foco del estudio en encontrar el modelo más adecuado que se amolde al tipo de datos y resultados que queremos alcanzar. Por ello el previo estudio de los modelos nos hace llegar a la idea de realizar una red neuronal del tipo LSTM que es la que más se ajusta a nuestro proyecto.

6.2 *Trabajo futuro*

Como trabajo futuro, se presenta la mejora en la automatización de la recolección de datos. Desde un principio el proyecto ha sido dependiente de la cantidad de datos recolectados. Por ello, se podría generar un modelo de automatización que cargara los datos (publicados en webs públicas) automáticamente, de esta manera nuestro modelo de predicción trabajaría dinámicamente proporcionando predicciones actualizadas en el tiempo. Además, a medida que se disponga de mayor cantidad de datos, mejorará el modelo ya que la calidad de las predicciones será mejor porque las redes de *Deep Learning* tendrán mayor conjunto de datos de entrenamiento.

Otra de las mejoras podría ser incluir otros factores relevantes en el estudio de la enfermedad, como datos de movilidad o climatológicos. A su vez, también podríamos ampliar el estudio con datos sanitarios excluyentes, como el impacto de la enfermedad en personas con enfermedades crónicas o estudiar la evolución de las personas vacunadas dependiendo de la marca de la vacuna suministrada. En nuestro caso queríamos evaluar por grupos de edad, pero existen múltiples variables en los que podemos generar grupos de población, como por ejemplo el género.

En este proyecto, hemos enfocado el estudio a los países de la Unión Europea ya que eran los datos más fiables con los que contábamos, pero podríamos extender a nivel mundial este estudio o incluso centrarnos simplemente en una región demográfica concreta.

En relación a la implementación de un modelo de *Machine Learning*, al tratarse de un amplio campo de conocimiento, se propone como trabajo futuro la ampliación de las clases para entrenar la red o la implementación de otro tipo de redes neuronales.

Referencias

[1] Chollet, F. (2018). “Deep Learning with Python”.

[2] Banco Mundial. <https://databank.bancomundial.org/source/world-development-indicators> [Consulta Febrero 2021]

[3] European Centre for Disease Prevention and Control <https://www.ecdc.europa.eu/en/publications-data/> [Consultas en primer cuatrimestre de 2021]

[4] Organización Mundial de la Salud. <https://covid19.who.int/table> [Consultas en primer cuatrimestre de 2021]

[5] Google <https://www.google.com/covid19/mobility/> [Consultas en primer cuatrimestre de 2021]

[6] Novac, I., Geipel, K., Gil, J., de Paula, L. y Hyttel, K. (2020). “A Framework for Wildfire Inspection Using Deep Convolutional Neural Networks”. IEEE/SICE INTERNATIONAL SYMPOSIUM ON SYSTEM INTEGRATION (SII) 2020, 0, 867-872.

[7] Van Roode, S. (2019). “An artificial neural network ensemble approach to generate air pollution maps. Environ. Monit. Assess”. 2019, 191, 727.

[8] Alonso, R. <https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/> [Artículo publicado el 08 de Abril de 2020]

[9] Calvo, D. <https://www.diegocalvo.es/perceptron-multicapa/> [Artículo publicado el 08 de Diciembre de 2018]

[10] Calvo, D. <https://www.diegocalvo.es/definicion-de-red-neuronal/> [Artículo publicado el 12 de Julio de 2017]

[11] <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/< [Artículo publicado el 29 de Noviembre de 2018]

[12] <https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html> [Consultas en primer cuatrimestre de 2021]

[13] Calvo, D. <https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/> [Artículo publicado el 13 de Julio de 2017]

[14] <https://blog.structuralia.com/python-inteligencia-artificial> [Artículo publicado el 23 de Octubre de 2020]

[15] Bisong, E. (2019). Google Colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform (pp. 59-64). Apress, Berkeley, CA.

[16] <https://colab.research.google.com/notebooks/welcome.ipynb?hl=es> [Consultas en primer cuatrimestre de 2021]

[17] <https://platzi.com/blog/librerias-de-machine-learning-tensorflow-scikit-learn-pythorch-y-keras/> [Artículo publicado en 2018]

[18] <https://aprendeia.com/introduccion-a-numpy-python-1/> [Artículo publicado el 21 de Septiembre de 2018]

[19] Amate, I. y Guarnido Rueda, A. (2010). "Factores determinantes del desarrollo económico y social"

[20] <http://pyciencia.blogspot.com/2015/04/trabajar-con-datos-nan-en-dataframe.html> [Artículo publicado el 03 de Abril de 2015]

[21] <http://educativa.catedu.es/44700165/aula/archivos/repositorio/2000/2010/html/22_la_distribucin_normal_estndar_y_su_tabla.html> [Consultas en primer cuatrimestre de 2021]

[22] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> [Consultas en primer cuatrimestre de 2021]

[23] <https://www.codificandobits.com/blog/redes-lstm/> [Artículo publicado el 20 de Julio de 2019]

[24] <https://matplotlib.org/stable/gallery/color/named_colors.html>[Consultas en primer cuatrimestre de 2021]

[25] <https://www.youtube.com/watch?v=00MDTu9HhpQ> [Vídeo publicado el 21 de Julio de 2020]

[26] <https://guru99.es/rnn-tutorial/> [Consultas en primer cuatrimestre de 2021]

[27] <https://es.wikipedia.org/wiki/Tangente_hiperbólica> [Consultas en primer cuatrimestre de 2021]

Glosario

CSV	C*omma-Separated Values*
DL	*Deep Learning*
ECDC	*European Centre for Disease Prevention and Control*
GHO	*Global Health Observatory*
GPU	*Graphics Processing Unit*
HTML	*HyperText Markup Language*
IA	Inteligencia artificial
IDE	*Integrated Development Environment*
IDH	Índice de Desarrollo Humano
LSTM	*Long short-term memory*
ML	*Machine Learning*
MSE	*Mean Squared Error*
NAN	*Not a Number*
OMS	Organización Mundial de la Salud
PCA	*Principal Component Analysis*
PIB	Producto Interior Bruto
PNUD	Programa de las Naciones Unidas para el Desarrollo
RNN	Red Neuronal Recurrente o *Recurrent Neural Network*
SPYDER	*Scientific Python Development Environment*
UE	Unión Europea
WB	*World Bank*

Anexos

A Gráficas resultados Red Neuronal. Predicciones España

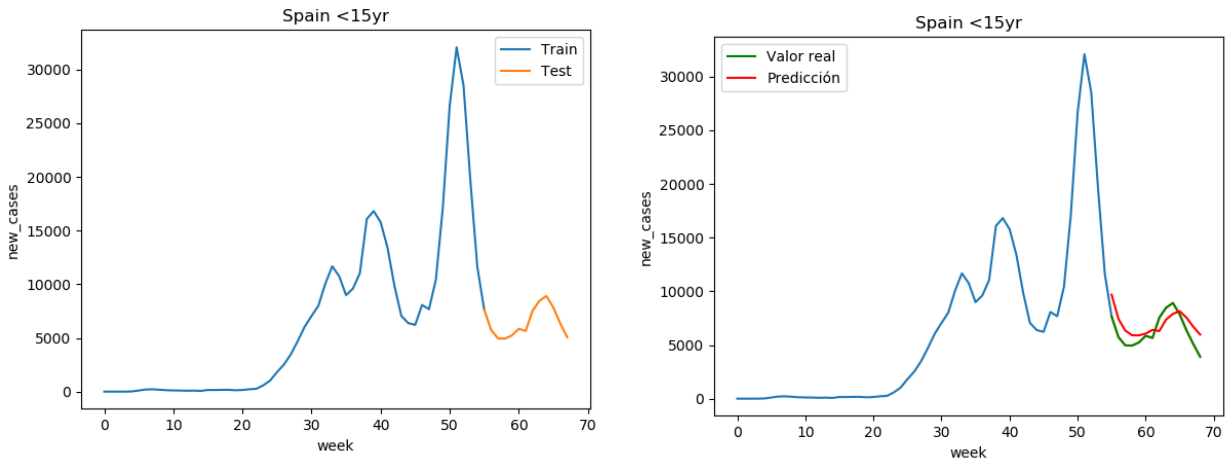


Figura 52 : Datos "Train" y "Test" y Predicción vs Valor real Spain <15yr. Fuente: Elaboración propia.

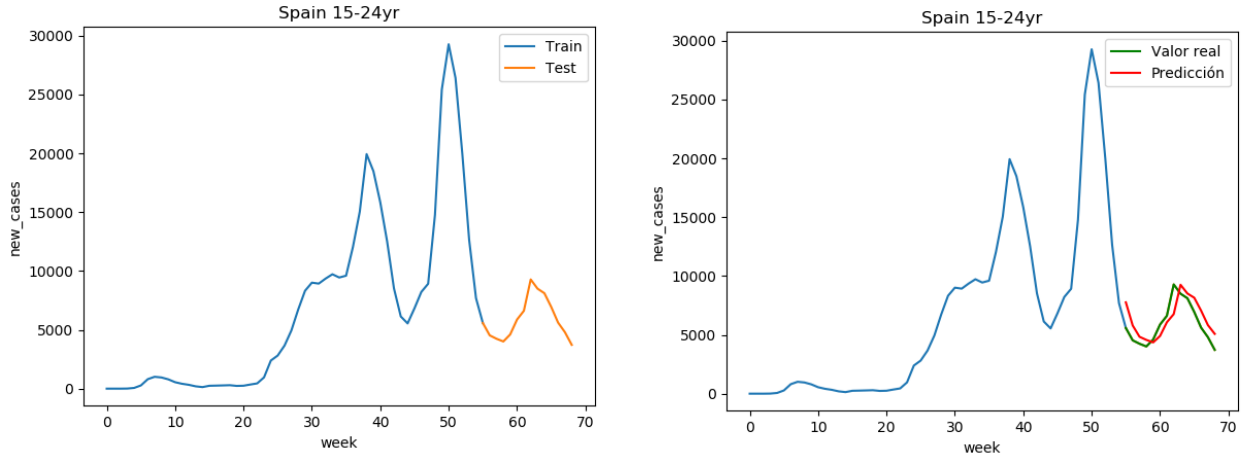


Figura 53 : Datos "Train" y "Test" y Predicción vs Valor real Spain 15-24yr. Fuente: Elaboración propia.

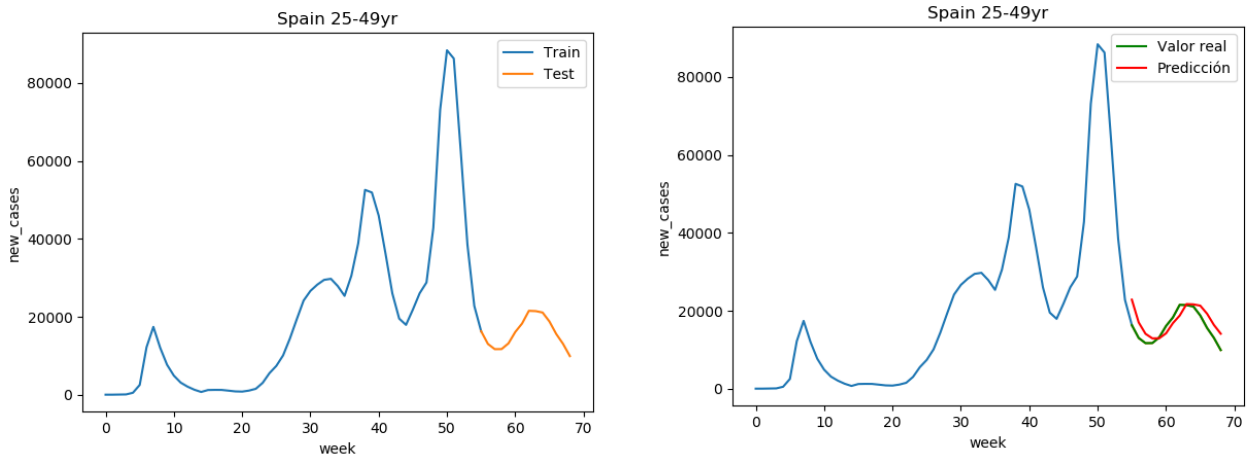


Figura 54 : Datos “Train” y “Test” y Predicción vs Valor real Spain 25-49yr. Fuente: Elaboración propia.

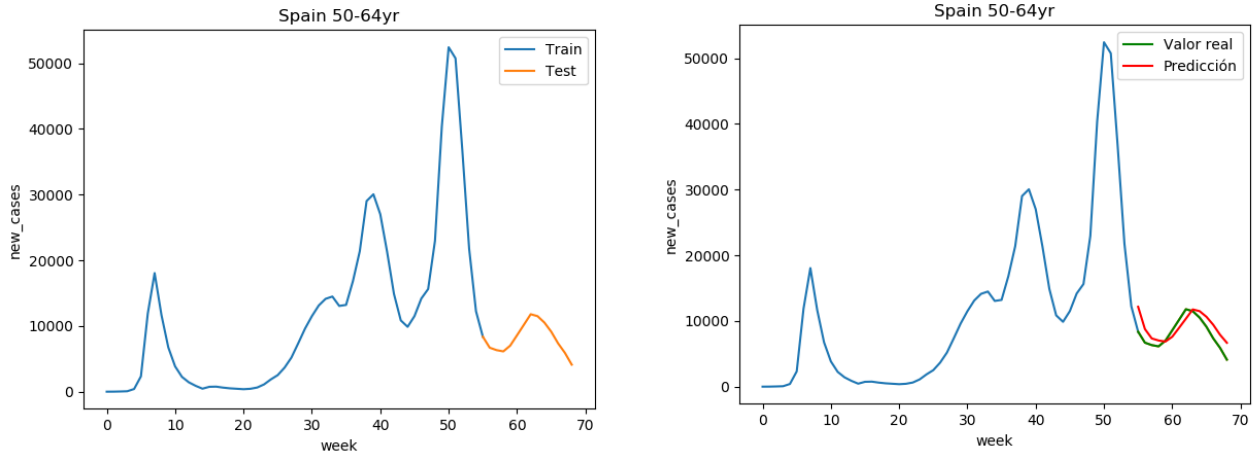


Figura 55 : Datos “Train” y “Test” y Predicción vs Valor real Spain 50-64yr. Fuente: Elaboración propia.

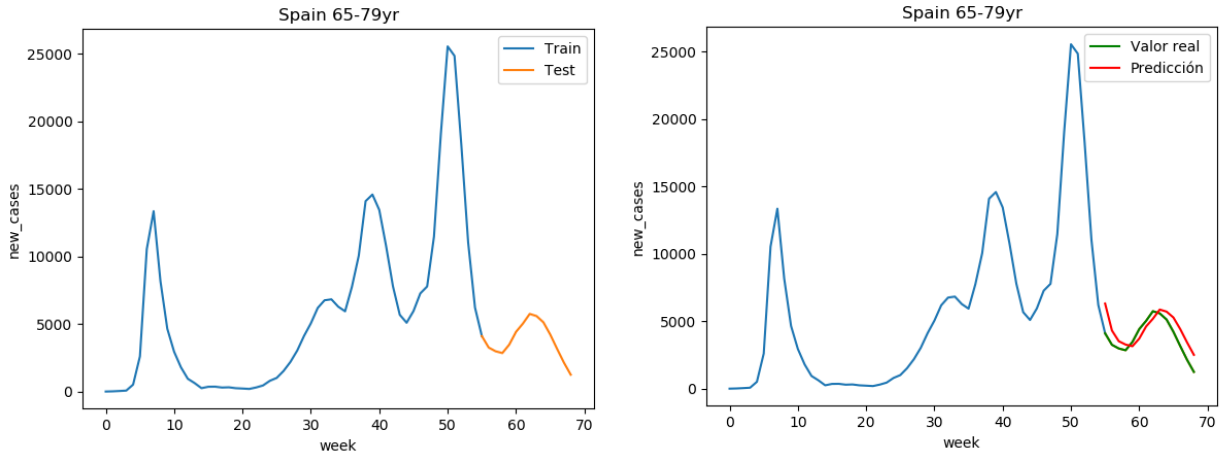


Figura 56 : Datos “Train” y “Test” y Predicción vs Valor real Spain 65-79yr. Fuente: Elaboración propia.

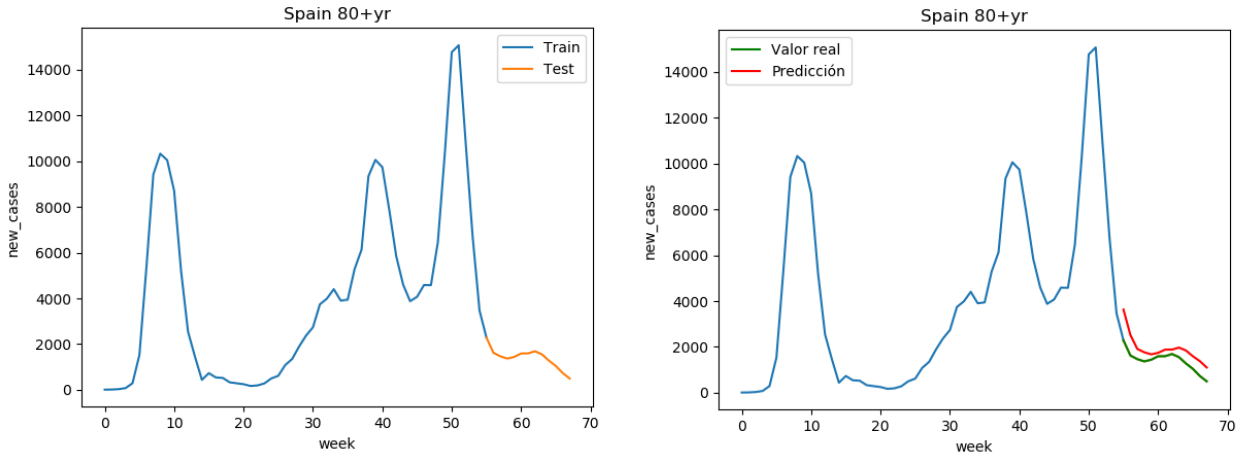


Figura 57 : Datos “Train” y “Test” y Predicción vs Valor real Spain 80+yr. Fuente: Elaboración propia.

B Gráficas países. Casos y Población

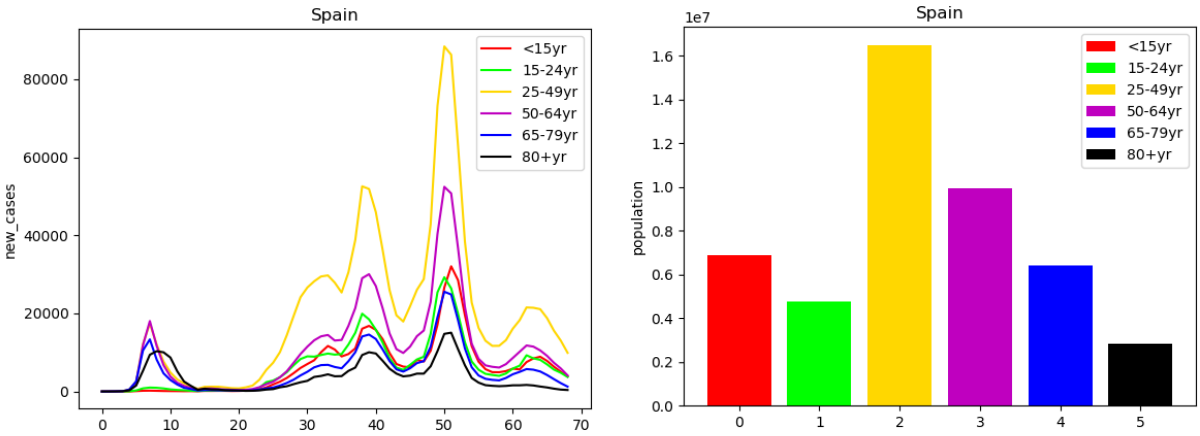


Figura 58 : Casos COVID y Pirámide población Spain. Fuente: Elaboración propia.

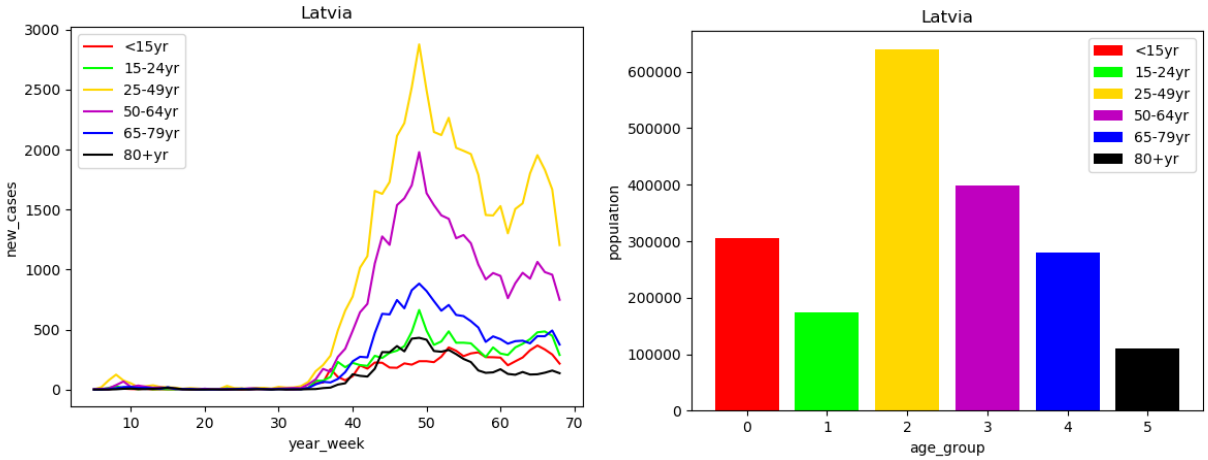


Figura 59 : Casos COVID y Pirámide población Latvia. Fuente: Elaboración propia.

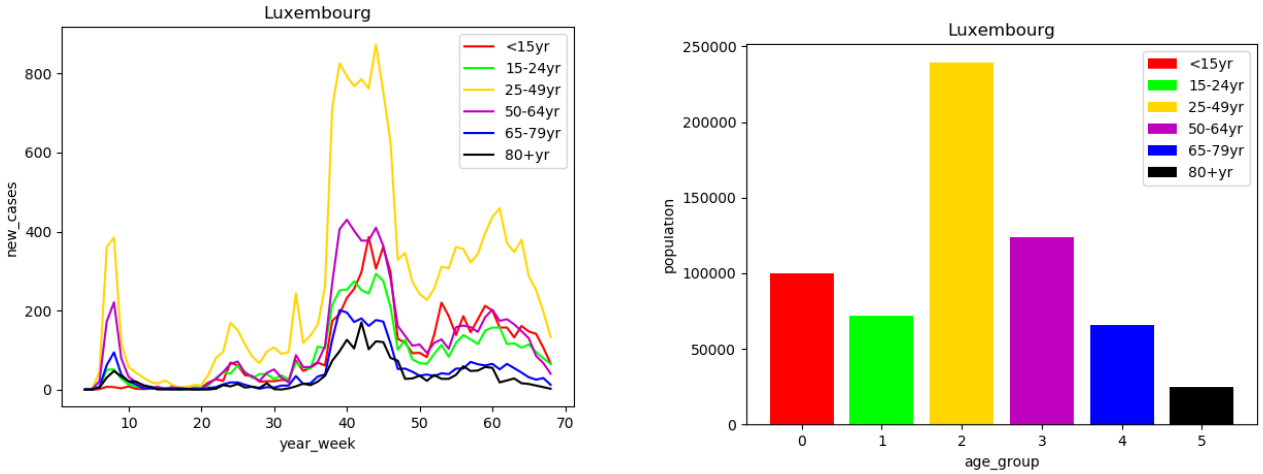


Figura 60: Casos COVID y Pirámide población Luxembourg. Fuente: Elaboración propia

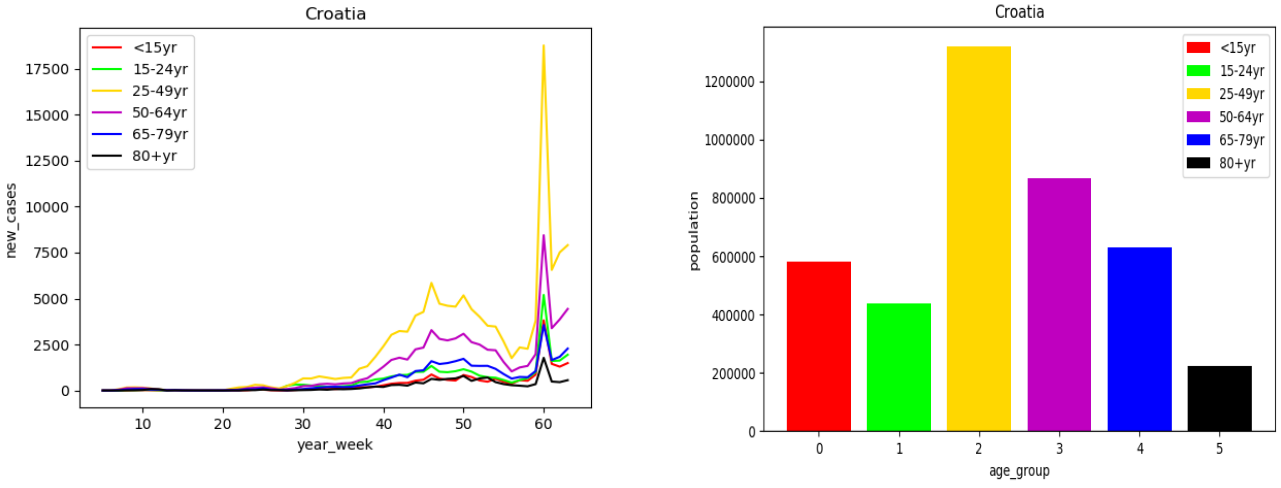


Figura 61: Casos COVID y Pirámide población Croatia. Fuente: Elaboración propia.

C Gráficas Vacunación España

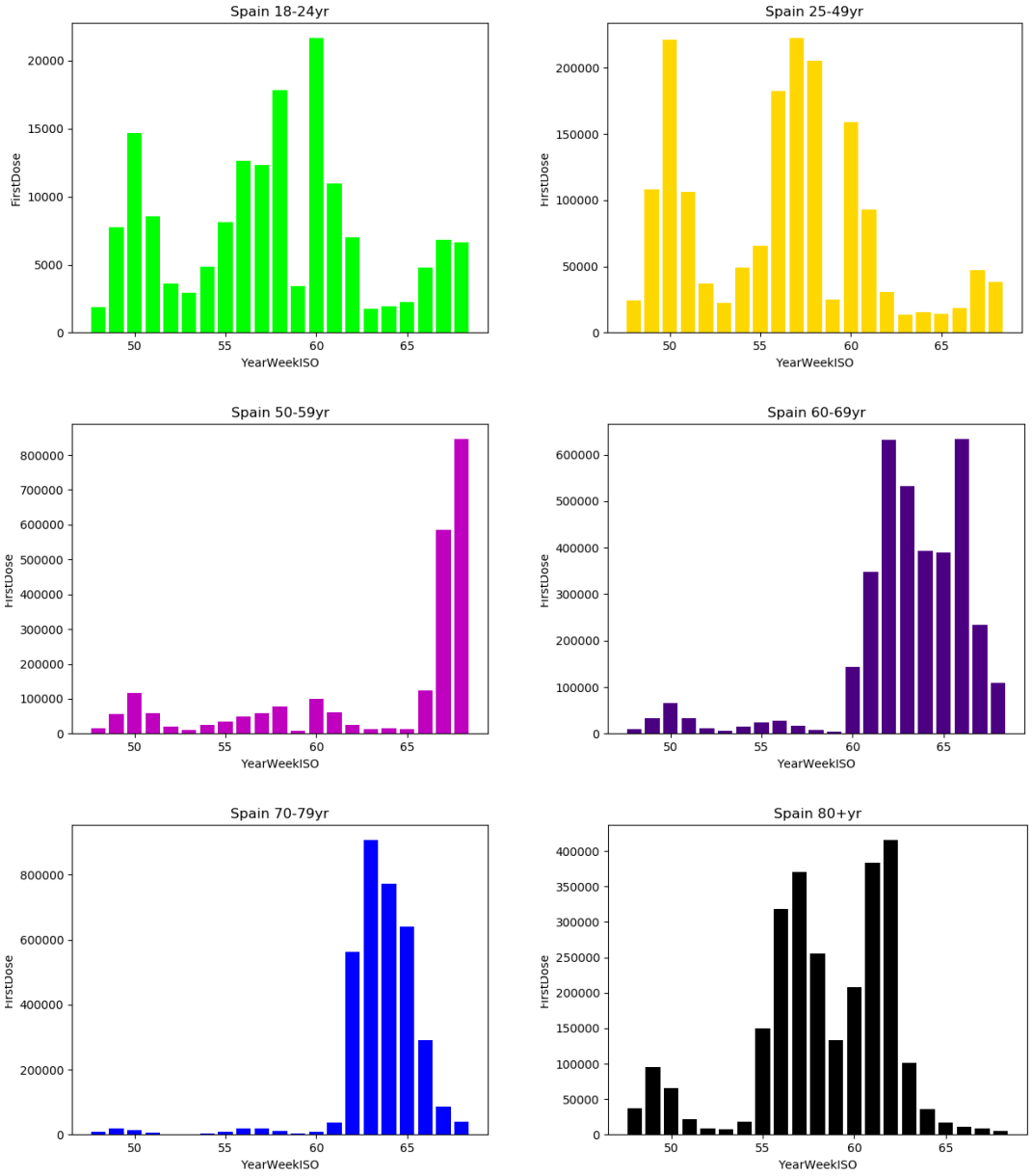


Figura 62 : Gráficas vacunación Spain 18-80+ yr. Fuente: Elaboración propia.

D Gráficas Vacunación Letonia

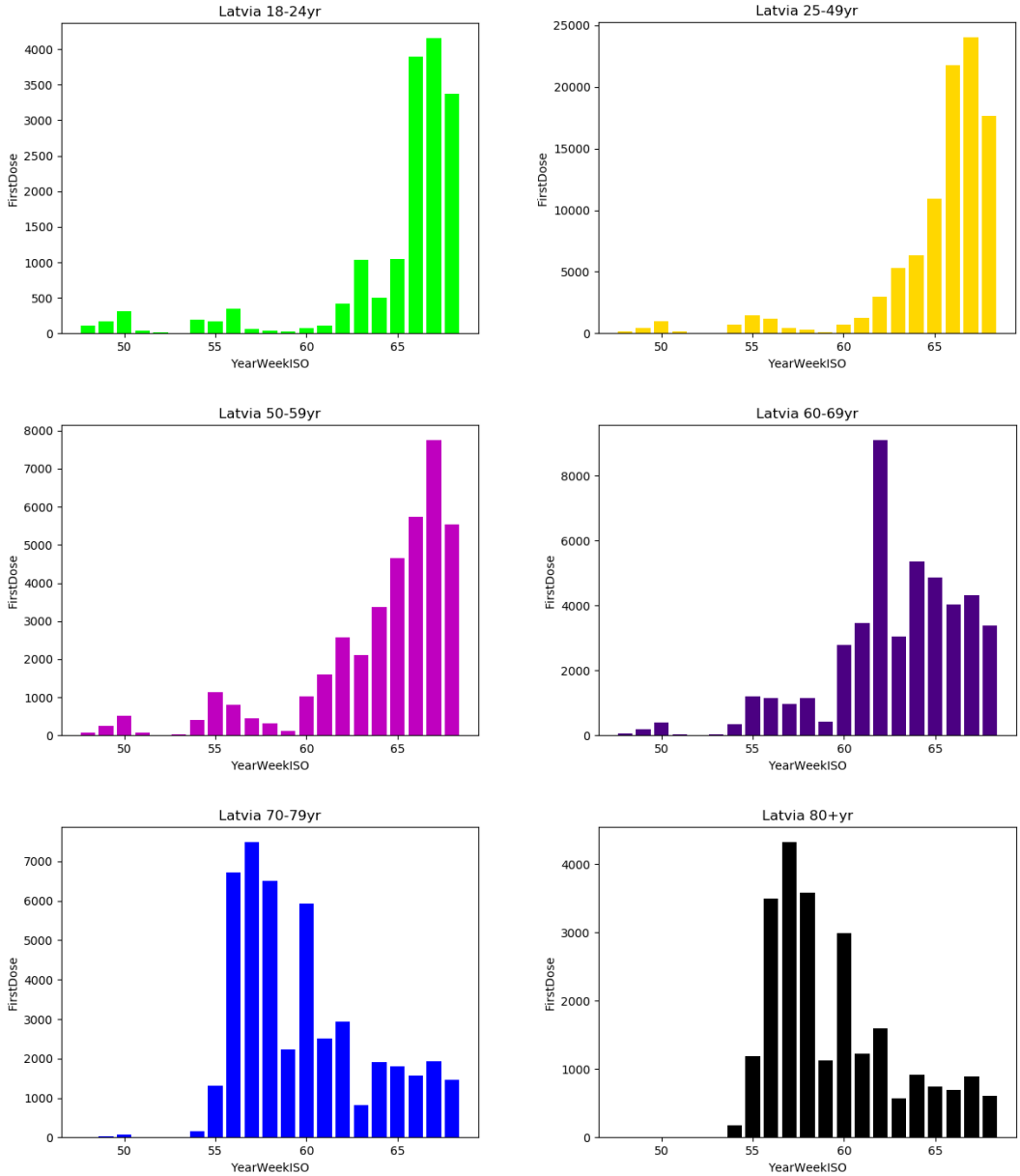


Figura 63 : Gráficas vacunación Latvia 18-80+ yr. Fuente: Elaboración propia.

E Gráficas Vacunación Luxemburgo

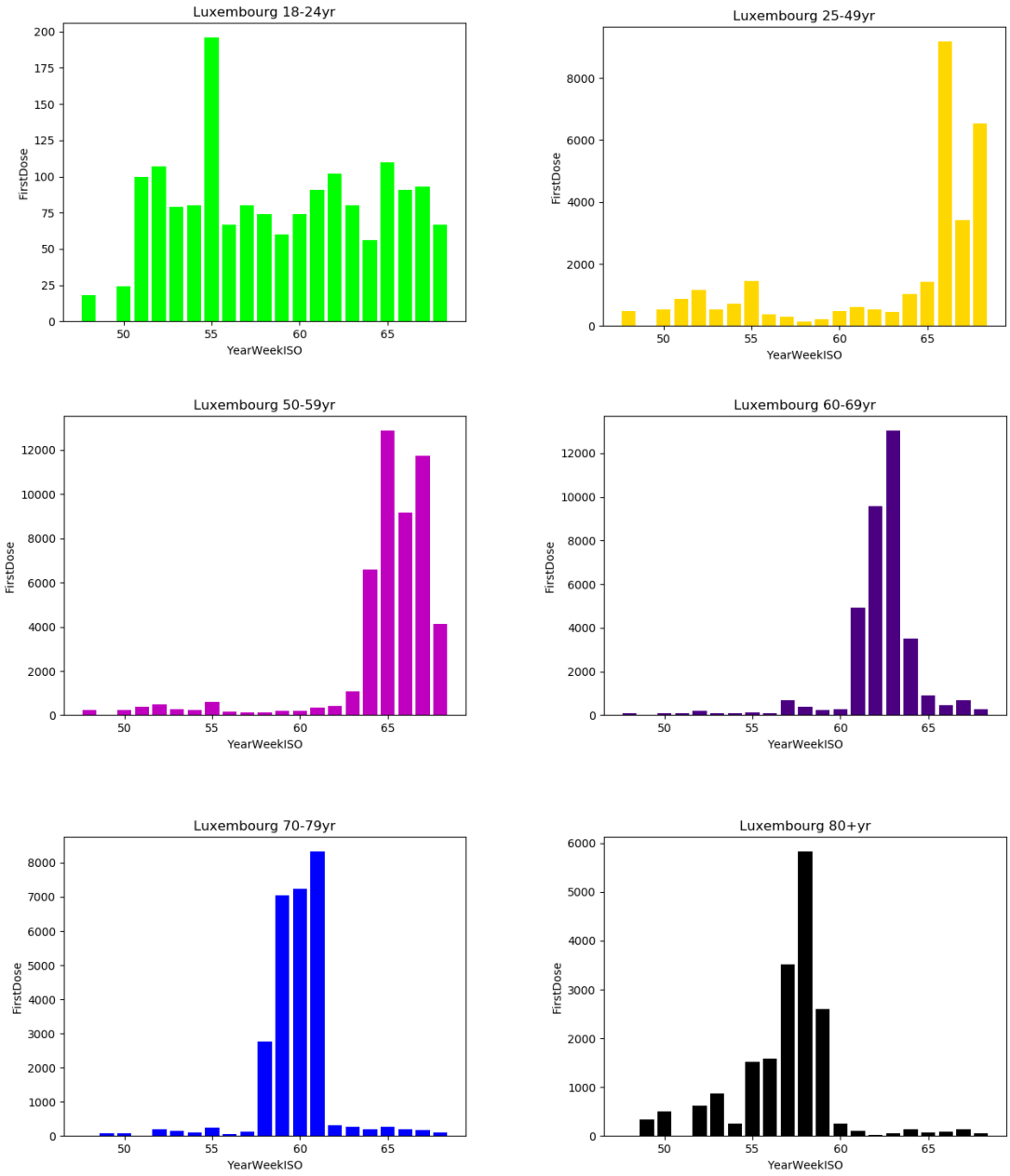


Figura 64 : Gráficas vacunación Luxembourg 18-80+ yr. Fuente: Elaboración propia.

F Gráficas Vacunación Croacia

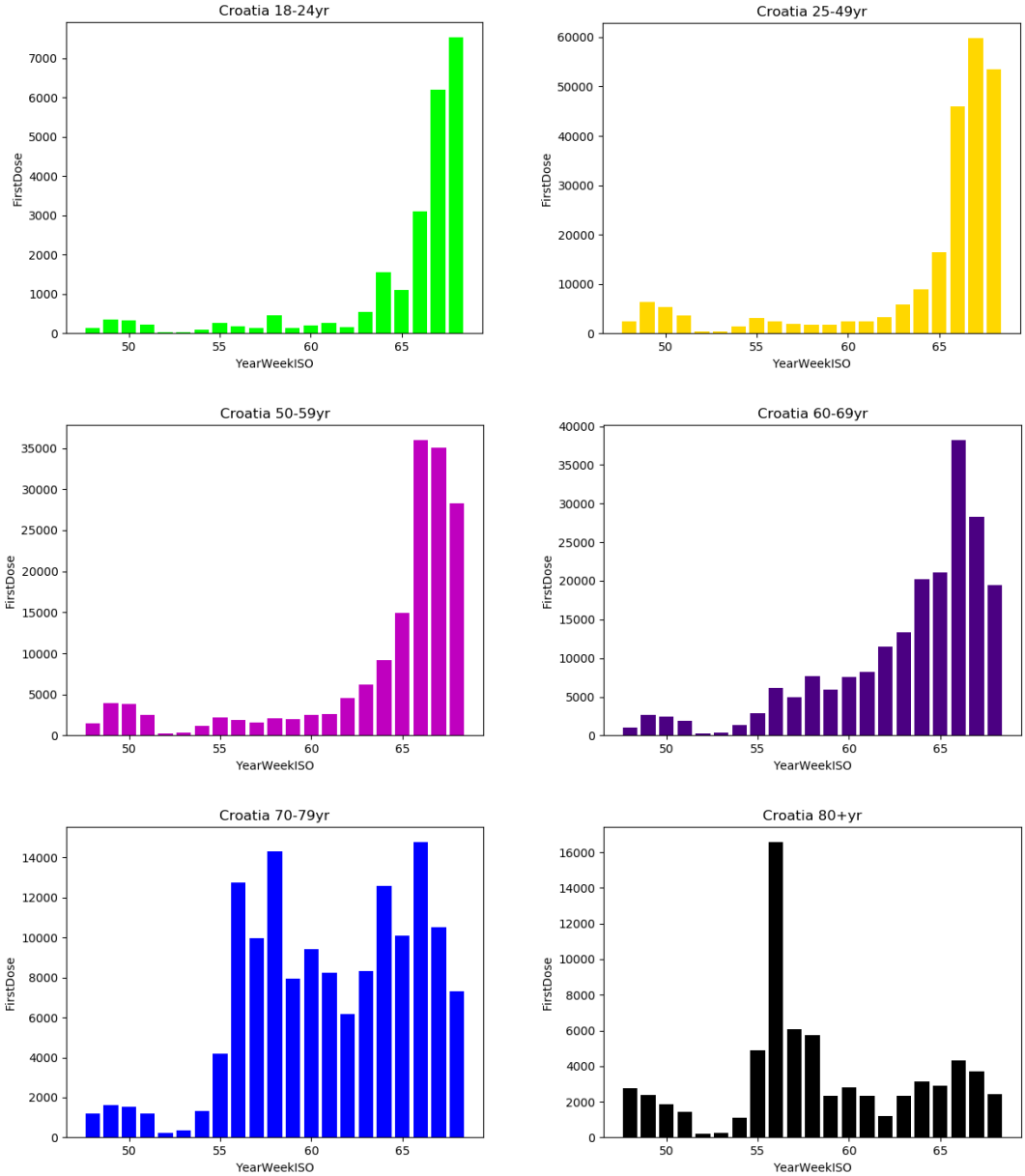


Figura 65 : Gráficas vacunación Croatia 18-80+ yr. Fuente: Elaboración propia.